

Computing Query Probability with Incidence Algebras

Nilesh Dalvi, Karl Schnaitter and Dan Suciu

ABSTRACT

We describe an algorithm for evaluating queries over probabilistic databases using incidence algebras. The queries we consider are unions of conjunctive queries, and the probabilistic database are tuple-independent structures. Our algorithm runs in PTIME, on a subset of queries called "safe" queries. The algorithm is very simple, and easy to implement in practice, yet it is highly non-obvious. The role played by the incidence algebras is that it allows us to avoid computing subqueries that are provably hard.

1. INTRODUCTION

In this paper we show how to use *incidence algebras* to evaluate *unions of conjunctive queries* over probabilistic databases. These queries correspond to the select-project-join-union fragment of the relational algebra, and they also correspond to *existential positive formulas* of First Order Logic. A probabilistic database, also referred to as a *probabilistic structure*, is a pair (\mathbf{A}, P) where $\mathbf{A} = (A, R_1^A, \dots, R_k^A)$ is first order structure over vocabulary R_1, \dots, R_k , and P is a function that associates to each tuple t in \mathbf{A} a number $P(t) \in [0, 1]$. A probabilistic structure defines a probability distribution on the set of substructures \mathbf{B} of \mathbf{A} by:

$$P_{\mathbf{A}}(\mathbf{B}) = \prod_{i=1}^k \left(\prod_{t \in R_i^{\mathbf{B}}} P(t) \times \prod_{t \in R_i^{\mathbf{A}} - R_i^{\mathbf{B}}} (1 - P(t)) \right) \quad (1)$$

We describe a simple, yet quite non-obvious algorithm for computing the probability of an existential, positive FO sentence Φ , $P_{\mathbf{A}}(\Phi)$ ¹, based on Mobius' inversion formula in incidence algebras. The algorithm runs in polynomial time in the size of \mathbf{A} . The algorithm only applies to certain sentences, called *safe* sentences, and is sound and complete in the following way. It is sound, in that it computes correctly the probability for each safe sentence, and it is complete in that, for every fixed unsafe sentence Φ , computing $P_{\mathbf{A}}(\Phi)$ is hard for #P, even when all probabilities in the input structure are 1/2 or 1. The algorithm is more general than, and

¹This is the marginal probability, defined as: $P_{\mathbf{A}}(\Phi) = \sum_{\mathbf{B}: \mathbf{B} \models \Phi} P_{\mathbf{A}}(\mathbf{B})$.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

significantly simpler than a previous algorithm for conjunctive sentences [5].

The need to identify tractable queries over probabilistic data has been addressed in several previous works [4, 6, 11, 10]. These works provide conditions for the tractability of queries without self-joins. The only exception is [5], which considers conjunctive queries with self joins. We extend those results to a larger class of queries, and at the same time provide a very simple algorithm. On the other hand, some of the earlier work is complimentary to ours, e.g., the results that consider the effects of functional dependencies [11].

Our results have applications to probabilistic inference on positive Boolean expressions [7]. For every tuple t in a structure \mathbf{A} , let X_t be a distinct Boolean variable. Every existential positive FO sentence Φ defines a positive DNF Boolean expression over the variables X_t , sometimes called *lineage expression*, whose probability is the same as $P_{\mathbf{A}}(\Phi)$. Our result can be used to classify the complexity of computing the probability of Positive DNF formulas defined by a fixed sentence Φ . For example, the two sentences²

$$\begin{aligned} \Phi_1 &= R(x), S(x, y) \vee S(x, y), T(y) \vee R(x), T(y) \\ \Phi_2 &= R(x), S(x, y) \vee S(x, y), T(y) \end{aligned}$$

define two classes of positive Boolean DNF expressions (lineages):

$$\begin{aligned} F_1 &= \bigvee_{a \in R, (a,b) \in S} X_a Y_{a,b} \vee \bigvee_{(a,b) \in S, b \in T} Y_{a,b}, Z_b \vee \bigvee_{a \in R, b \in S} X_a Y_b \\ F_2 &= \bigvee_{a \in R, (a,b) \in S} X_a Y_{a,b} \vee \bigvee_{(a,b) \in S, b \in T} Y_{a,b}, Z_b \end{aligned}$$

Our result implies that, for each such class of Boolean formulas, either all formulas in that class can be evaluated in PTIME in the size of the formula, or the complexity for that class is hard for #P, even if all probabilities are either 1/2 or 1; e.g. F_1 can be evaluated in PTIME using our algorithm, while F_2 is complete for #P.

The PTIME algorithm we present here relies in a critical way on an interesting connection between existential positive FO sentences and incidence algebras [13]. By using the Mobius inversion formula in incidence algebras we resolve a major difficulty of the evaluation problem: a sentence that is in PTIME may have a subexpression that is hard. This is illustrated by Φ_1 above, which is in PTIME, but has Φ_2 as a subexpression, which is hard; to evaluate Φ_1 one must avoid trying to evaluate Φ_2 . Our solution is to express $P(\Phi)$ using Mobius' inversion formula: subexpressions of Φ that have a Mobius value of zero do not contribute to $P(\Phi)$, and this allows us to compute $P(\Phi)$ without computing its hard subexpressions. The Mobius inversion formula corresponds to the

²We omit quantifiers and drop the conjunct they are clear from the context, e.g. $\Phi_2 = \exists x \exists y (R(x) \wedge S(x, y) \vee S(x, y) \wedge T(y))$.

inclusion/exclusion principle, which is used ubiquitously in probabilistic inference: the connection between the two in the context of probabilistic inference has already been recognized in [8]. However, to the best of our knowledge, ours is the first application of the full power of Mobius inversion formula for probability computation, by exploiting its ability to remove hard subexpressions from a computation.

Another distinguishing, and quite non-obvious aspect of our approach is that we apply our algorithm on the CNF, rather than the more commonly used DNF representation of existential, positive FO sentences. This departure from the common representation of existential, positive FO is necessary in order to handle correctly existential quantifiers.

Our algorithm is conceptually very simple, and relies on two techniques: the Mobius inversion formula (to remove Boolean connectives), and independence (to remove existential variables). In the last part of the paper, we make a strong claim: that using Mobius' inversion formula is a *necessary* technique for completeness. To support this claim we examine how other techniques commonly used today in probabilistic inference could be applied to the evaluation problem for existential positive FO sentences, and show that they cannot lead to a complete PTIME algorithm. Such common techniques are: independence, disjointness, and conditioning. In conditioning, one chooses a Boolean variable X , then computes $P(F) = P(F \mid X)P(X) + P(F \mid \neg X)(1 - P(X))$. We give a PTIME algorithm based on these three techniques, for existential positive FO sentences, where conditioning is performed on subformulas of Φ instead of Boolean variables. We prove that this algorithm is not complete. More precisely, we show a formula Φ (Fig. 2) that is computable in PTIME, but for which it is not possible to compute $P(\Phi)$ by using a combination of independence, disjointness, and conditioning on subformulas. On the other hand, we note that conditioning has certain practical advantages that are lost by Mobius' inversion formula: by repeated conditioning one can construct a Free Binary Decision Diagram [14], which has further applications beyond probabilistic inference. There seems to be no procedure to convert Mobius' inversion formula into FBDDs; in fact, we conjecture that the formula in Fig. 2 does not have an FBDD whose size is polynomial in that of the input structure.

In earlier work [4, 6] we have studied the evaluation of *conjunctive queries (sentences) without self-joins* on probabilistic structures. For this restricted language, the safe sentences are precisely the *hierarchical* queries, and the evaluation algorithm is very simple. This algorithm has been adopted and extended by several systems [1, 11]. In more recent work [5] we have removed the restriction on no self-joins, but the resulting algorithm turned out to be very complex and impractical. It relied on a large number of intermediate steps, whose completeness was never formally proven. Instead of Mobius' inversion function, it used a difficult technique called “erasers”, which corresponds to conditioning: as we show in this paper, conditioning does not lead to a complete algorithm when applied to the all existential positive sentences.

Finally, we mention that a different way to define classes of Boolean formulas has been studied in the context of the *constraint satisfaction problem* (CSP). Creignou, and Creignou and Hermann [3, 2] showed that the counting version of the CSP problem has a dichotomy into PTIME and #P-complete. These results are orthogonal to ours: they define the class of formulas by specifying the set of Boolean operators, such as and/or/not/majority/parity etc, and do not restrict the shape of the Boolean formula otherwise. As a consequence, the only class where counting is in PTIME is defined by affine operators: all classes of monotone formulas are hard. In contrast, in our classification there exist classes of formulas that are

in PTIME, for example the class defined by Φ_1 above.

The paper is organized as follows. We describe incidence algebras and their connection to existential, positive FO in Sec. 2, and describe how to use independence in Sec. 3. In Sec. 4 we describe ranking, a necessary technique to make things work. We give our complete algorithm in Sec. 5, and describe an incomplete algorithm based on conditioning in Sec. 6.

2. EXISTENTIAL POSITIVE FO AND INCIDENCE ALGEBRAS

We describe here the connection between positive FO and incidence algebras. We start with basic notations.

2.1 Existential Positive FO

Fix a vocabulary $\bar{R} = \{R_1, R_2, \dots\}$. A *conjunctive* sentence φ is sentence obtained from positive relational atoms using \wedge and \exists :

$$\varphi = \exists \bar{x}. (r_1 \wedge \dots \wedge r_k) \quad (2)$$

We allow the use of constants. $Var(\varphi) = \bar{x}$ denotes the set of variables in φ , and $Atoms(\varphi) = \{r_1, \dots, r_k\}$ the set of atoms. Consider the undirected graph where the nodes are $Atoms(\varphi)$ and edges are pairs (r_i, r_j) s.t. r_i, r_j have a common variable. A *component* of φ is a connected component in this graph. Each conjunctive sentence φ can be written as:

$$\varphi = \gamma_1 \wedge \dots \wedge \gamma_p$$

where each γ_i is a component; in particular, γ_i and γ_j do not share any common variables, when $i \neq j$.

A *disjunctive* sentence is an expression of the form:

$$\varphi' = \gamma'_1 \vee \dots \vee \gamma'_q$$

where each γ'_i is a single component.

An *existential, positive* sentence Φ is obtained from positive atoms using \wedge , \exists and \vee ; we will refer to it briefly as *positive sentence*. We write a positive sentence either in DNF or in CNF:

$$\Phi = \varphi_1 \vee \dots \vee \varphi_m \quad (3)$$

$$\Phi = \varphi'_1 \wedge \dots \wedge \varphi'_M \quad (4)$$

where φ_i are conjunctive sentences in the DNF (3), and φ'_i are disjunctive sentences in the CNF (4). The DNF can be rewritten into the CNF by:

$$\Phi = \bigvee_{i=1, m} \bigwedge_{j=1, p_i} \gamma_{ij} = \bigwedge_f \bigvee_i \gamma_{if(i)}$$

where f ranges over functions with domain $[m]$ s.t. $\forall i \in [m]$, $f(i) \in [p_i]$. This rewriting can increase the size of the expression exponentially³. Finally, we will often drop \exists and \wedge when clear from the context, as in the examples in Sec. 1.

A classic result by Sagiv and Yannakakis [12] gives a necessary and sufficient condition for a logical implication of positive sentences written in DNF: if $\Phi = \bigvee_i \varphi_i$ and $\Phi' = \bigvee_j \varphi'_j$, then:

$$\Phi \Rightarrow \Phi' \quad \text{iff} \quad \forall i. \exists j. \varphi_i \Rightarrow \varphi'_j \quad (5)$$

No analogous property holds for CNF: $R(x, a), S(a, z)$ logically implies $R(x, y), S(y, z)$ (where a is a constant), but $R(x, a) \not\Rightarrow R(x, y), S(y, z)$ and $S(a, z) \not\Rightarrow R(x, y), S(y, z)$. We show in Sec. 4 a rewriting technique that enforces such a property.

³Our algorithm runs in PTIME *data complexity*; we do not address the expression complexity in this paper.

2.2 Incidence Algebras

Next, we review the basic notions in incidence algebras following Stanley [13]. A finite *lattice* is a finite ordered set (\hat{L}, \leq) where every two elements $u, v \in \hat{L}$ have a least upper bound $u \vee v$ and a greatest lower bound $u \wedge v$, usually called *join* and *meet*. Since it is finite, it has a minimum and a maximum element, denoted $\hat{0}, \hat{1}$. We denote $L = \hat{L} - \{\hat{1}\}$ (departing from [13], where L denotes $\hat{L} - \{\hat{0}, \hat{1}\}$). L is a meet-semilattice. $\mathbf{R}^{\hat{L}}$ is a finite dimensional vector space whose elements are functions $f : \hat{L} \rightarrow \mathbf{R}$. Denote $(e_u)_{u \in \hat{L}}$ the canonical basis: $e_u(u) = 1, e_u(v) = 0$ for $v \neq u$. The *incidence algebra* $I(\hat{L})$ is the algebra⁴ of linear functions $t : \mathbf{R}^{\hat{L}} \rightarrow \mathbf{R}^{\hat{L}}$ that satisfy: for all $u \in \hat{L}$, $t(e_u)$ belongs to the subspace generated by $\{e_v \mid u \leq v\}$; multiplication in $I(\hat{L})$ is defined by function composition. Equivalently, $I(\hat{L})$ consists of all $|\hat{L}| \times |\hat{L}|$ matrices $(a_{uv})_{u, v \in \hat{L}}$ where the only non-zero elements are for $u \leq v$, and multiplication is matrix multiplication. In this paper, all we need are two elements of the incidence algebra: $\zeta \in I(\hat{L})$, defined as $\zeta(u, v) = 1$ for all $u \leq v$; and its inverse, the Mobius function $\mu : \{(u, v) \mid u, v \in \hat{L}, u \leq v\} \rightarrow \mathbf{Z}$, defined by:

$$\begin{aligned} \mu_{\hat{L}}(u, u) &= 1 \\ \mu_{\hat{L}}(u, v) &= - \sum_{w: u < w \leq v} \mu_{\hat{L}}(w, v) \end{aligned}$$

We drop the subscript and write μ when \hat{L} is clear from the context.

The Mobius inversion formula, which is the key piece of our algorithm, expresses the fact that if $g = \zeta(f)$, then $f = \mu(g)$. Namely: if a function g is defined as $g(v) = \sum_{u \leq v} f(u)$, then $f(v) = \sum_{u \leq v} \mu(u, v)g(u)$.

2.3 Their Connection

A *labeled lattice* is a triple $\hat{\mathbf{L}} = (\hat{L}, \leq, \lambda)$ where (\hat{L}, \leq) is a lattice and λ assigns to each element in $u \in \hat{L}$ a positive FO sentence $\lambda(u)$ s.t. $\lambda(u) \equiv \lambda(v)$ iff $u = v$.

DEFINITION 2.1. A *D-lattice* is a labeled lattice $\hat{\mathbf{L}}$ where, for all $u \neq \hat{1}$, $\lambda(u)$ is conjunctive, for all u, v , $\lambda(u \wedge v)$ is logically equivalent to $\lambda(u) \wedge \lambda(v)$, and $\lambda(\hat{1}) \equiv \bigvee_{u < \hat{1}} \lambda(u)$.

A *C-lattice* is a labeled lattice $\hat{\mathbf{L}}$ where, for all $u \neq \hat{1}$, $\lambda(u)$ is disjunctive, for all u, v , $\lambda(u \wedge v)$ is logically equivalent to $\lambda(u) \vee \lambda(v)$, and $\lambda(\hat{1}) = \bigwedge_{u < \hat{1}} \lambda(u)$.

In a D-lattice, $u \leq v$ iff $\lambda(u) \Rightarrow \lambda(v)$. This is because $\lambda(u) = \lambda(u \wedge v)$ is logically equivalent to $\lambda(u) \wedge \lambda(v)$. Similarly, in a C-lattice, $u \leq v$ iff $\lambda(v) \Rightarrow \lambda(u)$. If $\hat{\mathbf{L}}$ is a D- or C-lattice, we say $\hat{\mathbf{L}}$ represents $\Phi = \lambda(\hat{1})$.

PROPOSITION 2.2 (INVERSION FORMULA FOR POSITIVE FO). Fix a probabilistic structure (\mathbf{A}, P) and a positive sentence Φ ; denote $P_{\mathbf{A}}$ as P . Let $\hat{\mathbf{L}}$ be either a D-lattice or a C-lattice representing Φ . Then:

$$P(\Phi) = P(\lambda(\hat{1})) = - \sum_{v < \hat{1}} \mu_L(v, \hat{1}) P(\lambda(v)) \quad (6)$$

PROOF. The proof for the D-lattice is from [13]. Denote $f(u) = P(\lambda(u) \wedge \neg(\bigvee_{v < u} \lambda(v)))$. Then:

$$P(\lambda(u)) = \sum_{v \leq u} f(v) \Rightarrow f(u) = \sum_{v \leq u} \mu(v, u) P(\lambda(v))$$

⁴An algebra is a vector space plus a multiplication operation [13].

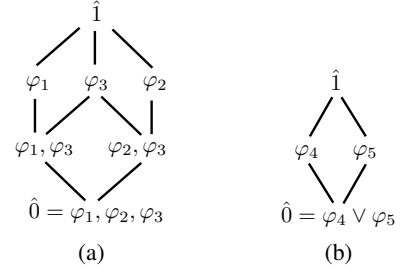


Figure 1: The D-lattice (a) and the C-lattice (b) for Φ (Ex. 2.3).

The claim follows by setting $u = \hat{1}$ and noting $f(\hat{1}) = 0$. For a C-lattice, write $\lambda'(u) = \neg\lambda(u)$. Then $P(\lambda(\hat{1})) = 1 - P(\lambda'(\hat{1})) = 1 + \sum_{v < \hat{1}} \mu(v, \hat{1}) P(\lambda'(v))$ and the claim follows from the fact that $\sum_{v \in \hat{L}} \mu(v, \hat{1}) = 0$. \square

The proposition generalizes the well known inclusion/exclusion formula (for D-lattices), and its less well known dual (for C-lattices):

$$\begin{aligned} P(a \vee b \vee c) &= P(a) + P(b) + P(c) \\ &\quad - P(a \wedge b) - P(a \wedge c) - P(b \wedge c) + P(a \wedge b \wedge c) \\ P(a \wedge b \wedge c) &= P(a) + P(b) + P(c) \\ &\quad - P(a \vee b) - P(a \vee c) - P(b \vee c) + P(a \vee b \vee c) \end{aligned}$$

We show how to construct a canonical D-lattice, $\hat{\mathbf{L}}_D(\Phi)$ that represents a positive sentence Φ . Start from the DNF in Eq.(3), and for each subset $s \subseteq [m]$ denote $\varphi_s = \bigwedge_{i \in s} \varphi_i$. Let \hat{L} be the set of these conjunctive sentences, up to logical equivalence, and ordered by logical implication (hence, $|\hat{L}| \leq 2^m$). Label each element $u \in \hat{L}$, $u \neq \hat{1}$, with its corresponding φ_s (choose any, if there are multiple equivalent ones), and label $\hat{1}$ with $\bigvee_{s \neq \emptyset} \varphi_s$ ($\equiv \Phi$). We denote the resulting D-lattice $\hat{\mathbf{L}}_D(\Phi)$. Similarly, $\hat{\mathbf{L}}_C(\Phi)$ is the C-lattice that represents Φ , obtained from the CNF of Φ in Eq.(4), setting $\varphi'_s = \bigvee_{i \in s} \varphi'_i$.

In summary, the first main technique of our algorithm is this. Given Φ , compute its C-lattice, then use Eq.(6) to compute $P(\Phi)$; we explain later why we use the C-lattice instead of the D-lattice. This reduces the problem to that of computing the probability of disjunctive sentences $P(\lambda(u))$: we show in the next section how to compute the latter. The power of this technique comes from the fact that, whenever $\mu(u, \hat{1}) = 0$, then we do not need to compute the corresponding $P(\lambda(u))$. As we explain in Sec. 6 this is strictly more powerful than the current techniques used in probabilistic inference, which are based on conditioning, independence, and disjointness.

Example 2.3 Consider the following positive sentence:

$$\begin{aligned} \Phi &= R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2) \vee R(x_3), T(y_3) \\ &= \varphi_1 \vee \varphi_2 \vee \varphi_3 \end{aligned}$$

The Hasse diagram of the D-lattice $\mathbf{L}_D(\Phi)$ is shown in Fig. 1 (a). There are eight subsets $s \subseteq [3]$, but only seven up to logical equivalence, because⁵ $\varphi_1, \varphi_2 \equiv \varphi_1, \varphi_2, \varphi_3$. The values of the Mobius function are, from top to bottom: 1, -1, -1, -1, 1, 1, 0, hence the inversion formula is:

$$P(\Phi) = P(\varphi_1) + P(\varphi_2) + P(\varphi_3) - P(\varphi_1 \varphi_3) - P(\varphi_2 \varphi_3)$$

⁵There exists a homomorphism $\varphi_1, \varphi_2, \varphi_3 \rightarrow \varphi_1, \varphi_2$ that maps $R(x_3)$ to $R(x_1)$ and $T(y_3)$ to $T(y_2)$.

The Hasse diagram of the C-lattice $\mathbf{L}_C(\Phi)$ is shown in Fig. 1 (b). To see this, first express Φ in CNF:

$$\begin{aligned}\Phi &= (R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2) \vee R(x_3)) \wedge \\ &\quad (R(x_4), S(x_4, y_4) \vee S(x_5, y_5), T(y_5) \vee T(y_6)) \\ &= (R(x_3) \vee S(x_2, y_2), T(y_2)) \wedge (R(x_4), S(x_4, y_4) \vee T(y_6)) \\ &= \varphi_4 \wedge \varphi_5\end{aligned}$$

Note that $\hat{0}$ is labeled with $\varphi_4 \vee \varphi_5 \equiv R(x_3) \vee T(y_6)$. The inversion formula here is:

$$P(\Phi) = P(\varphi_4) + P(\varphi_5) - P(\varphi_4 \vee \varphi_5)$$

where $\varphi_4 \vee \varphi_5 \equiv R(x_3) \vee T(y_6)$.

In general, there may be many lattices that represent the same positive sentence Φ . For example, consider any two conjunctive sentences s.t. $\varphi_2 \Rightarrow \varphi_1$, then φ_1 and $\varphi_1 \vee \varphi_2$ are equivalent positive sentences, yet their canonical D-lattices differ. One expects the algorithm to be invariant to equivalent expressions. We show this formally next. An element u in a lattice *covers* v if $u > v$ and there is no w s.t. $u > w > v$. An *atom*⁶ is an element that covers $\hat{0}$; a *co-atom* is an element covered by $\hat{1}$. An element u is called *co-atomic* if it is a meet of coatoms. Let L_0 denote the set of co-atomic elements: L_0 is a meet semilattice, and $\hat{L}_0 = L_0 \cup \{\hat{1}\}$ is a lattice. We prove the following in the Appendix:

PROPOSITION 2.4. (1) If $u \in L$ and $\mu_{\hat{L}}(u, \hat{1}) \neq 0$ then u is co-atomic. (2) For all $u \in L_0$, $\mu_{\hat{L}}(u, \hat{1}) = \mu_{\hat{L}_0}(u, \hat{1})$.

Let \hat{L} and \hat{L}' be D-lattices representing the sentences Φ and Φ' . If $\Phi \equiv \Phi'$, then \hat{L} and \hat{L}' have the same co-atoms, up to logical equivalence. Indeed, we can write Φ as the disjunction of co-atom labels in \hat{L} , and one co-atom cannot imply another. Thus, by applying Eq.(5) in both directions, we get a one-to-one correspondence between the co-atoms of \hat{L} and \hat{L}' , indicating logical equivalence. It follows from Prop. 2.4 that, when D-lattices represent equivalent formulas, the set of labels $\lambda(u)$ where $\mu(u, \hat{1}) \neq 0$ are equivalent. Thus, an algorithm that inspects only these labels is independent of the particular representation of a sentence.

A similar property does not hold for C-lattices, because Eq.(5) does not extend to CNF. For example, $\Phi = R(x, a), S(a, z)$ and $\Phi' = R(x, a), S(a, z), R(x', y'), S(y', z')$ are logically equivalent, but have different co-atoms. The co-atoms of Φ are $R(x, a)$ and $S(a, z)$ (the C-lattice is V-shaped, as in Fig. 1 (b)), and the co-atoms of Φ' are $R(x, a)$, $(R(x', y'), S(y', z'))$, and $S(a, z)$ (the C-lattice is W-shaped, as in Fig. 1 (a)). However, we prove in Sec. 4.2 that over ranked structures, C-lattices representing equivalent formulas have the same sets of co-atoms.

3. INDEPENDENCE AND SEPARATORS

Next, we show how to compute the probability of a disjunctive sentence $\bigvee_i \gamma_i$; this is the second technique used in our algorithm, and consists of eliminating, simultaneously, one existential variable from each γ_i , by exploiting independence.

Let φ be a conjunctive sentence. A *valuation* h is a substitution of its variables with constants; $h(\varphi)$, is a set of ground tuples. We call two conjunctive sentences φ_1, φ_2 *tuple-independent* if for all valuations h_1, h_2 , we have $h_1(\varphi_1) \cap h_2(\varphi_2) = \emptyset$. Two positive sentences Φ, Φ' are *tuple-independent* if, after expressing them in DNF, $\Phi = \bigvee_i \varphi_i, \Phi' = \bigvee_j \varphi'_j$, all pairs φ_i, φ'_j are independent.

⁶Not to be confused with a relational atom r_i in (2).

Let Φ_1, \dots, Φ_m be positive sentences s.t. any two are tuple-independent. Then:

$$P(\bigvee_i \Phi_i) = 1 - \prod_i (1 - P(\Phi_i))$$

This is because the m lineage expressions for Φ_i depend on disjoint sets of Boolean variables, and therefore they are independent probabilistic events. In other words, tuple-independence is a sufficient condition for independence in the probabilistic sense. Although it is only a sufficient condition, we will abbreviate tuple-independence with independence in this section.

Let φ be a positive sentence, $V = \{x_1, \dots, x_m\} \subseteq \text{Vars}(\varphi)$, and a a constant. Denote $\varphi[a/V] = \varphi[a/x_1, \dots, a/x_m]$ (all variables in V are substituted with a).

DEFINITION 3.1. Let $\varphi = \bigvee_{i=1,m} \gamma_i$ be a disjunctive sentence. A separator is a set of variables $V = \{x_1, \dots, x_m\}$, $x_i \in \text{Var}(\gamma_i)$, such that for all $a \neq b$, $\varphi[a/V], \varphi[b/V]$ are independent.

PROPOSITION 3.2. Let φ be a disjunctive sentence with a separator V , and (\mathbf{A}, P) a probabilistic structure with active domain D . Then:

$$P(\varphi) = 1 - \prod_{a \in D} (1 - P(\varphi[a/V])) \quad (7)$$

The claim follows from the fact that $\varphi \equiv \bigvee_{a \in D} \varphi[a/V]$ on all structures whose active domain is included in D .

In summary, to compute the probability of a disjunctive sentence, we find a separator, then apply Eq.(7): each expression $\varphi[a/V]$ is a positive sentence, simpler than the original one (it has strictly fewer variables in each atom) and we apply again the inversion formula. This technique, by itself, is not complete: we need to “rank” the relations in order to make it complete, as we show in the next section. Before that, we illustrate with an example.

Example 3.3 Consider $\varphi = R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(x_2)$. Here $\{x_1, x_2\}$ is a separator. To see this, note that for any constants $a \neq b$, the sentences $\varphi[a] = R(a), S(a, y_1) \vee S(a, y_2), T(a)$ and $\varphi[b] = R(b), S(b, y_1) \vee S(b, y_2), T(b)$ are independent, because the former only looks at tuples that start with a , while the latter only looks at tuples that start with b .

Consider $\varphi = R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2)$. This sentence has no separator. For example, $\{x_1, x_2\}$ is not a separator because both sentences $\varphi[a]$ and $\varphi[b]$ have the atom $T(y_2)$ in common: if two homomorphisms h_1, h_2 map y_2 to some constant c , then $T(c) \in h_1(\varphi[a]) \cap h_2(\varphi[b])$, hence they are dependent. The set $\{x_1, y_2\}$ is also not a separator, because $\varphi[a]$ contains the atom $S(a, y_1)$, $\varphi[b]$ contains the atom $S(x_2, b)$, and these two can be mapped to the common ground tuple $S(a, b)$.

We end with a necessary condition for V to be a separator.

DEFINITION 3.4. If γ is a component, a variable of γ is called a root variable if it occurs in all atoms of γ .

Note that components do not necessarily have root variables, e.g., $R(x), S(x, y), T(y)$. We have:

PROPOSITION 3.5. If V is a separator of $\bigvee_i \gamma_i$, then each separator variable $x_i \in \text{Vars}(\gamma_i)$ is a root variable for γ_i .

The claim follows from the fact that, if r is any atom in φ_i that does not contain x_i : then r is unchanged in $\gamma_i[a]$ and in $\gamma_i[b]$, hence they are not independent.

4. RANKING

In this section, we define a simple restriction on all formulas and structures that simplify our later analysis: we require that, in each relation, the attributes are strictly ordered $A_1 < A_2 < \dots$. We show how to alter any positive sentence and probabilistic structure to satisfy these constraints, without changing the sentence probability. This is a necessary preprocessing step for our algorithm to work, and a very convenient technique in the proofs.

4.1 Ranked Structures

Let C be a fixed set of constants: later we will choose C to be the set of constants used in a given sentence Φ .

DEFINITION 4.1. A relation R is ranked w.r.t. C if every tuple $R(a_1, \dots, a_k)$ is such that $a_1 < \dots < a_k$ and $a_i \notin C$, for $i = 1, k$. A probabilistic structure is ranked w.r.t. C if all its relations are ranked.

To motivate ranked structures, we observe that the techniques given in previous sections do not directly lead to a complete algorithm. For example, the sentence $\gamma = R(x, y), R(y, x)$ is connected, so we cannot use Mobius inversion to simplify it. We also cannot apply Eq.(7) because there is no separator: indeed, $\{x\}$ is not a separator because $R(a, y), R(y, a)$ and $R(b, y), R(y, b)$ are not independent (they share the tuple $R(a, b)$), and by symmetry neither is $\{y\}$. However, consider a structure with a unary relation R_{12} and binary relations $R_{1<2}, R_{2<1}$ defined as:

$$R_{12} = \pi_{X_1}(\sigma_{X_1=X_2}(R)) \quad R_{2<1} = \pi_{X_2 X_1}(\sigma_{X_2 < X_1}(R)) \\ R_{1<2} = \sigma_{X_1 < X_2}(R)$$

Here, we use X_i to refer to the i -th attribute of R . This is a ranked structure: in both relations $R_{1<2}$ and $R_{2<1}$ the first attribute is less than the second. Moreover: $\gamma \equiv R_{12}(z) \vee R_{1<2}(x, y), R_{2<1}(x, y)$ and now $\{z, x\}$ is a separator, because $R_{1<2}(a, y), R_{2<1}(a, y)$ and $R_{1<2}(b, y), R_{2<1}(b, y)$ are independent. Thus, Eq.(7) applies to the formula over the ranked structure, and we can compute the probability of γ in polynomial time.

Once we restrict the structures to be ranked, we will remove all constants and duplicate variables in the atoms of a sentence; note that an atom with duplicate variables cannot be satisfied by a ranked relation, nor can an atom with a constant be satisfied, since we assume that the structure is ranked w.r.t. all the constants that originally occurred in the sentence.

DEFINITION 4.2. We say a positive sentence is in reduced form if each atom $R(x_1, \dots, x_k)$ is such that each x_i is a distinct variable.

We now prove that the evaluation of any sentence can be reduced to an equivalent sentence over a ranked structure, and we further guarantee that the resulting sentence is in reduced form.

PROPOSITION 4.3. Let Φ_0 be positive sentence and let C be the set of constants used in Φ_0 . For any structure \mathbf{A}_0 , there exists structure \mathbf{A} that is ranked w.r.t. C , and a sentence Φ in reduced form, such that $P_{\mathbf{A}_0}(\Phi_0) = P_{\mathbf{A}}(\Phi)$.

PROOF. Let $R(X_1, \dots, X_k)$ be a relation symbol and let ρ be a maximal, consistent conjunction of order predicates involving attributes of R and the constants occurring in Φ_0 : for any attributes or constants y, z , ρ implies exactly one of $y < z, y = z, y > z$. We say X_j is unbound if $\rho \not\Rightarrow X_j = c$ for any constant c . We denote $R^\rho = \pi_{\bar{X}}(\sigma_\rho(R))$ where \bar{X} contains one X_j in each class of unbound attributes that are equivalent under ρ , listed in increasing order according to ρ .

We show how to rewrite any positive sentence into an equivalent, reduced sentence over ranked structures. We start with a conjunctive sentence $\varphi = r_1, \dots, r_n$ and let R_i denote the relation symbol of r_i . Consider a maximally consistent predicate ρ_i on the attributes of R_i , for each $i = 1, n$, and let $\rho \equiv \rho_1, \dots, \rho_n$ be the conjunction. We say that ρ is consistent if there is a valuation h such that $h(\varphi) \models \rho$. Given a consistent ρ , divide the variables into equivalence classes of variables that ρ requires to be equal, and choose one representative variable from each class. Let $r_i^{\rho_i}$ be the result of changing $R_i(x_1, \dots, x_k)$ to $R_i^{\rho_i}(y_1, \dots, y_m)$, where y_1, \dots, y_m are chosen as follows. Consider the unbound attribute classes in R_i , in increasing order according to ρ_i . Choose y_p to be the representative of a variable that occurs in the position of an attribute in the p -th class of unbound attributes. This works because the position of any unbound attribute X must have a variable: if there is a constant a , then $h(r_i) \models X = a$ for all valuations h . But $\rho_i \Rightarrow X \neq a$ so this contradicts the assumption that ρ is consistent. Using a similar argument, we can show that each y_i is distinct, so $r_i^{\rho_i}$ is in reduced form. Furthermore, $\varphi \equiv \bigwedge_\rho r_1^{\rho_1}, \dots, r_n^{\rho_n}$ where the disjunction ranges over all maximal ρ_i such that ρ is consistent. For a positive sentence Φ_0 , we apply the above procedure to each conjunctive sentence in the DNF of Φ_0 to yield a sentence in reduced form on the ranked relations R^ρ . \square

Example 4.4 Let $\varphi = R(x, a), R(a, x)$. If we define the ranked relations $R_1 = \pi_{X_2}(\sigma_{X_1=a}(R))$, $R_2 = \pi_{X_1}(\sigma_{X_2=a}(R))$, and $R_{12} = \pi_\emptyset(\sigma_{X_1=X_2=a}(R))$, we have $\varphi \equiv R_1(x), R_2(x) \vee R_{12}()$.

Next, consider $\varphi = R(x), S(x, x, y), S(u, v, v)$. Define

$$S_{123} = \pi_{X_1}(\sigma_{X_1=X_2=X_3}(S)) \\ S_{23<1} = \pi_{X_2 X_1}(\sigma_{X_2=X_3 < X_1}((S)))$$

and so on. We can rewrite φ as:

$$\varphi \equiv R(x), S_{123}(x) \\ \vee R(x), S_{12<3}(x, y), S_{1<23}(u, v) \vee R(x), S_{12<3}(x, y), S_{23<1}(v, u) \\ \vee R(x), S_{3<12}(y, x), S_{1<23}(u, v) \vee R(x), S_{3<12}(y, x), S_{23<1}(v, u)$$

and note that these relations are ranked. \square

Thus, when computing $P_{\mathbf{A}}(\Phi)$, we may conveniently assume w.l.o.g. that \mathbf{A} is ranked and Φ is in reduced form. When we replace separator variables with a constant as in Eq.(7), we can easily restore the formula to reduced form. Given a disjunctive sentence φ in reduced form and a separator V , we remove a from $\varphi[a/V]$ as follows. For each relation R , suppose the separator variables occur at position X_i of R . Then we remove all rows from R where $X_i \neq a$, reduce the arity of R by removing column i , and remove $x_i = a$ from all atoms $R(x_1, \dots, x_k)$ in $\varphi[a/V]$.

We end this section with two applications of ranking. The first shows a homomorphism theorem for CNF sentences.

PROPOSITION 4.5. Assume all structures to be ranked, and all sentences to be in reduced form.

- If φ, φ' are conjunctive sentences, and φ is satisfiable over ranked structures⁷ then $\varphi \Rightarrow \varphi'$ iff there exists a homomorphism $h : \varphi' \rightarrow \varphi$.
- Formula (5) holds for positive sentences in DNF.
- The dual of (5) holds for positive sentences in CNF:

$$\bigwedge_i \varphi_i \Rightarrow \bigwedge_j \varphi'_j \quad \text{iff} \quad \forall j. \exists i. \varphi_i \Rightarrow \varphi'_j$$

⁷Meaning: it is satisfied by at least one (ranked) structure.

The proof is in the appendix. The first two items are known to fail for conjunctive sentences with order predicates: for example $R(x, y)$, $R(y, x)$ logically implies $R(x, y), x \leq y$, but there is no homomorphism from the latter to the former. They hold for ranked structures because there is a strict total order on the attributes of each relation. The last item implies the following. If $\hat{\mathbf{L}}$ and $\hat{\mathbf{L}}'$ are two C-lattices representing equivalent sentences, then they have the same co-atoms. In conjunction with Prop. 2.4, this implies that an algorithm that ignores lattice elements where $\mu(u, \hat{1}) = 0$ does not depend on the representation of the positive sentence. This completes our discussion at the end of Sec. 2.

The second result shows how to handle atoms without variables.

PROPOSITION 4.6. *Let γ_0, γ_1 be components in reduced form s.t. $\text{Var}(\gamma_0) = \emptyset$, $\text{Var}(\gamma_1) \neq \emptyset$. Then γ_0, γ_1 are independent.*

PROOF. Note that γ_0 contains a single atom $R()$; if it had two atoms then it is not a component. Since γ_1 is connected, each atom must have at least one variable, hence it cannot have the same relation symbol $R()$. \square

Let $\varphi = \bigvee \gamma_i$ be a disjunctive sentence, $\varphi_0 = \bigvee_{i: \text{Var}(\gamma_i) = \emptyset} \gamma_i$ and $\varphi_1 = \bigvee_{i: \text{Var}(\gamma_i) \neq \emptyset} \gamma_i$. It follows that:

$$P(\varphi) = 1 - (1 - P(\varphi_0))(1 - P(\varphi_1)) \quad (8)$$

4.2 Finding a Separator

Assuming structures to be ranked, we give here a necessary and sufficient condition for a disjunctive sentence in reduced form to have a separator, which we use both in the algorithm and to prove hardness for #P. We need some definitions first.

Let $\varphi = \gamma_1 \vee \dots \vee \gamma_m$ be a disjunctive sentence, in reduced form. Throughout this section we assume that φ is minimized: more precisely each γ_i is minimized, and is non-redundant (there is no γ_j s.t. $\gamma_i \Rightarrow \gamma_j$). This representation of φ is unique up to isomorphism. Further assume $\text{Var}(\gamma_i) \cap \text{Var}(\gamma_j) = \emptyset$ for all $i \neq j$ (if not, then rename the variables). Two atoms $r \in \text{Atoms}(\gamma_i)$ and $r' \in \text{Atoms}(\gamma_j)$ are called *unifiable* if they have the same relational symbol. We may also say r, r' *unify*. It is easy to see that γ_i and γ_j contain two unifiable atoms iff they are not tuple-independent. Two variables x, x' are *unifiable* if there exist two unifiable atoms r, r' such that x occurs in r at the same position that x' occurs in r' . This relationship is reflexive and symmetric. We also say that x, x' are *recursively unifiable* if either x, x' are unifiable, or there exists a variable x'' such that x, x'' and x', x'' are recursively unifiable.

A variable x is *maximal* if it is only recursively unifiable with root variables. Hence all maximal variables are root variables. The following are canonical examples of sentences where each component has a root variable, but there are no maximal variables:

$$\begin{aligned} h_0 &= R(x_0), S_1(x_0, y_0), T(y_0) \\ h_1 &= R(x_0), S_1(x_0, y_0) \vee S_1(x_1, y_1), T(y_1) \\ h_2 &= R(x_0), S_1(x_0, y_0) \vee S_1(x_1, y_1), S_2(x_1, y_1) \vee S_2(x_2, y_2), T(y_2) \\ &\dots \\ h_k &= R(x_0), S_1(x_0, y_0) \vee S_1(x_1, y_1), S_2(x_1, y_1) \vee \\ &\dots \vee S_{k-1}(x_{k-1}, y_{k-1}), S_k(x_{k-1}, y_{k-1}) \vee (S_k(x_k, y_k), T(y_k)) \end{aligned}$$

In each h_k the root variables are x_{i-1}, y_i for $i = 1, k-1$, and there are no maximal variables.

Maximality propagates during unification: if x is maximal and x, x' unify, then x' must be maximal because otherwise x would recursively unify with a non-root variable.

Let W_i be the set of maximal variables occurring in γ_i . If an atom in γ_i unifies with an atom in γ_j , then $|W_i| = |W_j|$ because the two atoms contain all maximal variables in each component,

Algorithm 5.1 Algorithm for Computing $P(\Phi)$

Input : Positive sentence Φ in reduced form;

Ranked structure (\mathbf{A}, p) with active domain D

Output : $P(\Phi)$

```

1: Function MobiusStep( $\Phi$ ) /*  $\Phi$  = positive sentence */
2:   Let  $\hat{\mathbf{L}} = \hat{\mathbf{L}}_C(\Phi)$  be a C-lattice representing  $\Phi$ 
3:   Return  $\sum_{u < \hat{1}} \mu_{\hat{\mathbf{L}}}(u, \hat{1}) * \text{IndepStep}(\lambda(u))$ 
4:    $\square$ 
5: Function IndepStep( $\varphi$ ) /*  $\varphi = \bigvee_i \gamma_i$  */
6:   Remove redundant  $\gamma_i$ 's, i.e. for which  $\exists j \neq i$  s.t.  $\gamma_i \Rightarrow \gamma_j$ 
7:   Minimize each component  $\gamma_i$ .
8:   Let  $\varphi = \varphi_0 \vee \varphi_1$ 
9:   where:  $\varphi_0 = \bigvee_{i: \text{Var}(\gamma_i) = \emptyset} \gamma_i$ ,  $\varphi_1 = \bigvee_{i: \text{Var}(\gamma_i) \neq \emptyset} \gamma_i$ 
10:  Let  $V$  = a separator for  $\varphi_1$  (Sec. 4.2)
11:  If (no separator exists) then FAIL (UNSAFE)
12:  Let  $p_0 = P(\varphi_0)$ 
13:  Let  $p_1 = 1 - \prod_{a \in D} (1 - \text{MobiusStep}(\varphi_1[a/V]))$ 
14:  /* Note: assume  $\varphi_1[a/V]$  is reduced (Sec.4) */
15:  Return  $1 - (1 - p_0)(1 - p_1)$ .
16:   $\square$ 

```

and maximality propagates through unification. Since the structures are ranked, for every i there exists a total order on the maximal variables in W_i : $x_{i1} < x_{i2} < \dots$. The *rank* of a variable $x \in W_i$ is the position where it occurs in this order. The following result gives us a means to find a separator if it exists:

PROPOSITION 4.7. *A disjunctive sentence has a separator iff every component has a maximal variable. In that case, the set comprising maximal variables with rank 1 forms a separator.*

PROOF. Consider the disjunctive sentence $\varphi = \bigvee_{i=1}^m \gamma_i$ and set of variables $V = \{x_1, \dots, x_m\}$ s.t. $x_i \in \text{Vars}(\gamma_i)$, $i = 1, m$. It is straightforward to show that V is a separator iff any pair of unifiable atoms have a member of V in the same position. Hence, if V is a separator, then each $x_i \in V$ can only (recursively) unify with another $x_j \in V$. Since x_j is a root variable (Prop. 3.5), each $x_i \in \text{Vars}(\gamma_i)$ is maximal, as desired.

Now suppose every component has a maximal variable. Choose V such that x_i is the maximal variable in γ_i with rank 1. If two atoms r, r' unify, then they have maximal variables occurring in the same positions. In particular, the first maximal variable has rank 1, and thus is in V . We conclude that V is a separator. \square

For a trivial illustration of this result, consider the disjunctive sentence $R(x, y), S(x, y) \vee S(x', y'), T(x', y')$. All variables are root variables, and the sets of maximal variables are $W_1 = \{x, y\}$, $W_2 = \{x', y'\}$. We break the tie by using the ranking: choosing arbitrarily rank 1, we obtain the separator $\{x, x'\}$. (Rank 2 would give us the separator $\{y, y'\}$). A more interesting example is:

Example 4.8 In φ , not all root variables are maximal:

$$\varphi = R(z_1, x_1), S(z_1, x_1, y_1) \vee S(z_2, x_2, y_2), T(z_2, y_2) \vee R(z_3, x_3), T(z_3, y_3)$$

The root variables are z_1, x_1, z_2, y_2, z_3 . The sets of maximal variables in each component are $W_1 = \{z_1\}$, $W_2 = \{z_2\}$, $W_3 = \{z_3\}$, and the set $\{z_1, z_2, z_3\}$ is a separator.

5. THE ALGORITHM

Algorithm 5.1 takes as input a ranked probabilistic structure \mathbf{A} and a positive sentence Φ in reduced form, and computes the probability $P(\Phi)$, or fails. The algorithm proceeds recursively on the

structure of the sentence Φ . The first step applies the Mobius inversion formula Eq.(6) to the C-lattice for Φ , expressing $P(\Phi)$ as a sum of several $P(\varphi)$, where each φ is a disjunctive sentence. Skipping those φ 's where the Mobius function is zero, for all others it proceeds with the second step. Here, the algorithm first “minimizes” φ , by removing redundant components and by minimizing all remaining components. A component γ_i is redundant if there exists another component s.t. $\gamma_i \Rightarrow \gamma_j$. Minimizing γ_i means replacing it with a smallest subset of its atoms, γ'_i , s.t. there exists a homomorphism $\gamma_i \rightarrow \gamma'_i$ (γ'_i is sometimes called the *core* of γ_i). Finally, compute $P(\bigvee \gamma_i)$, by using Eq.(8), and Eq.(7). For the latter, the algorithm needs to find a separator first, as described in Sec. 4.2: if none exists, then the algorithm fails.

The expression $P(\varphi_0)$ represents the base case of the algorithm: this is when the recursion stops, when all variables have been substituted with constants from the structure \mathbf{A} . Notice that φ_0 is of the form $\bigvee r_i$, where each r_i is a ground atom. Its probability is $1 - \prod_i (1 - P(r_i))$, where P is the probability function of the probabilistic structure (\mathbf{A}, P) .

We will now define *safe* sentences Φ , on which the algorithm always succeeds, and for that we need some definitions. Let φ be a disjunctive sentence. A *level* is a non-empty set of variables⁸ W such that every atom in φ contains at most one variable in W and for any unifiable variables x, x' , if $x \in W$ then $x' \in W$. Note that, in particular, any separator is a level. For a variable $x \in W$, let n_x be the number of atoms that contain x ; let $n = \max_x n_x$. Let $A = \{a_1, \dots, a_k\}$ be a set of constants not occurring in φ s.t. $k \leq n$. Denote $\varphi[A/W]$ the sentence obtained as follows: substitute each variable $x \in W$ with some constant $a_i \in A$ and take the union of all such substitutions:

$$\varphi[A/W] = \bigvee_{\theta: W \rightarrow A} \varphi[\theta]$$

Note that $\varphi[A/W]$ is not necessarily a disjunctive sentence, since some components γ_i may become disconnected in $\varphi[A/W]$.

DEFINITION 5.1. Define the following rewrite rule $\Phi \rightarrow \Phi_0$ on positive sentences. Below, $\varphi, \varphi_0, \varphi_1$, denote disjunctive sentences:

$$\begin{array}{lll} \varphi & \rightarrow & \varphi[A/W] \quad W \text{ is a level, } A \text{ is a set of constants} \\ \varphi_0 \vee \varphi_1 & \rightarrow & \varphi_1 \quad \text{if } \text{Vars}(\varphi_0) = \emptyset \\ \Phi & \rightarrow & \varphi \quad \exists u \in \mathbf{L}_C(\Phi). \mu(u, \hat{1}) \neq 0, \varphi = \lambda(u) \end{array}$$

The second and third rules are called *simple rules*. The first rule is also *simple* if W is a separator and $|A| = 1$.

The first rewrite rule allows us to substitute variables with constants; the second to get rid of disjuncts without any variables; the last rule allows us to replace a CNF sentence Φ with one element of its C-lattice, provided its Mobius value is non-zero.

DEFINITION 5.2. A positive sentence Φ is called *unsafe* if there exists a sequence of simple rewritings $\Phi \xrightarrow{*} \varphi$ s.t. φ is a disjunctive sentence without separators. Otherwise it is called *safe*.

Let Φ be an FO sentence. The *weak counting problem* for Φ is the following. Given a structure \mathbf{A} and a substructure \mathbf{A}_0 , compute the number of structures \mathbf{B} s.t. $\mathbf{A}_0 \subseteq \mathbf{B} \subseteq \mathbf{A}$ and $\mathbf{B} \models \Phi$. Equivalently, compute $P(\Phi)$ on the probabilistic structure \mathbf{A} where all tuples in \mathbf{A}_0 have probability 1 and all other tuples have probability 1/2. The main result in this paper is:

THEOREM 5.3 (SOUNDNESS AND COMPLETENESS). Fix a positive sentence Φ .

⁸No connection to the maximal sets W_i in Sec. 4.2.

Soundness If Φ is safe then, for any probabilistic structure, Algorithm 5.1 terminates successfully (i.e. doesn't fail), computes correctly $P(\Phi)$, and runs in time $O(n^k)$, where n is the size of the active domain in the structure, and k the largest arity of any symbol in the vocabulary.

Completeness If Φ is unsafe then the weak counting problem for Φ is hard for $\#P$.

Soundness follows immediately, by induction: if the algorithms starts with Φ , then for any sentence Φ_0 processed recursively, it is the case that $\Phi \xrightarrow{*} \Phi_0$, where all rewrites are simple. Thus, if the algorithm ever gets stuck, Φ is unsafe. The complexity follows from the fact that each recursive step of the algorithm removes one variable from every atom, and traverses the domain D once, at a cost $O(n)$. We discuss completeness in Sec. 5.2.

Example 5.4 Let $\Phi = R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2) \vee R(x_3), T(y_3)$. This example is interesting because the subexpression $R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2)$ is $\#P$ -hard (it has no separator), but the entire sentence is in PTIME. The algorithm computes the C-lattice, shown in Fig. 1 (b), then expresses $P(\Phi) = P(\varphi_4) + P(\varphi_5) - P(\varphi_6)$ where $\varphi_6 = R(x) \vee T(y)$ (see Example 2.3 for notations). Next, the algorithm applies the independence step to each of $\varphi_4, \varphi_5, \varphi_6$; we illustrate here for $\varphi_4 = R(x_3) \vee S(x_2, y_2), T(y_2)$ only; the other expressions are similar. Here, $\{x_3, y_2\}$ is a set of separator variables, hence:

$$P(\varphi_4) = 1 - \prod_{a \in A} (1 - P(R(a) \vee S(x_2, a), T(a)))$$

Next, we apply the algorithm recursively on $R(a) \vee S(x_2, a), T(a)$. In CNF it becomes⁹ $(R(a) \vee S(x_2, a))(R(a) \vee T(a))$, and the algorithm returns $P(R(a) \vee S(x_2, a)) + P(R(a) \vee T(a)) - P(R(a) \vee S(x_2, a) \vee T(a))$. Consider the last of the three expressions (the other two are similar): its probability is

$$1 - (1 - P(R(a) \vee T(a))) \prod_{b \in A} (1 - P(S(b, a)))$$

Now we have finally reached the base case, where we compute the probabilities of sentences without variables: $P(R(a) \vee T(a)) = 1 - (1 - P(R(a)))(1 - P(T(a)))$, and similarly for the others.

Example 5.5 Consider the sentence φ in Example 4.8. Since this is already CNF (it is a disjunctive sentence), the algorithm proceeds directly to the second step. The separator is $V = \{z_1, z_2, z_3\}$ (see Ex. 4.8), and therefore:

$$P(\varphi) = 1 - \prod_{a \in A} (1 - P(\varphi[a/V]))$$

where $\varphi[a/V]$ is:

$$R(a, x_1), S(a, x_1, y_1) \vee S(a, x_2, y_2), T(a, y_2) \vee R(a, x_3), T(a, y_3)$$

After reducing the sentence (i.e. removing the constant a), it becomes identical to Example 5.4.

5.1 Discussion

We justify here two major choices we made in the algorithm: using the C-lattice instead of the D-lattice, and relying on the inversion formula with the Mobius function instead of some simpler method to eliminate unsafe subexpressions.

⁹Strictly speaking, we would have had to rewrite the sentence into a reduce form first, by rewriting $S(x_2, a)$ into $S_{2 < a}(x_2)$, etc.

To see the need for the C-lattice, let's examine a possible dual algorithm, which applies the Mobius step to the D-lattice. Such an algorithm fails on Ex. 4.8, because here the D-lattice is $2^{[3]}$, and the Mobius function is $+1$ or -1 for every element of the lattice. The lattice contains $R(z_1, x_1), S(z_1, x_1, y_1), S(z_2, x_2, y_2), T(z_2, y_2)$, which is unsafe¹⁰. Thus, the dual algorithm fails.

To see the need of the Mobius inversion, we prove that an existential, positive FO sentence can be “as hard as any lattice”.

THEOREM 5.6 (REPRESENTATION THEOREM). *Let (\hat{L}, \leq) be any lattice. There exists a positive sentence Φ such that: $\mathbf{L}_D(\Phi) = (\hat{L}, \leq, \lambda)$, $\lambda(\hat{0})$ is unsafe, and for all $u \neq \hat{0}$, $\lambda(u)$ is safe. The dual statement holds for the C-lattice.*

PROOF. Call an element $r \in L$ join irreducible if whenever $v_1 \vee v_2 = r$, then either $v_1 = r$ or $v_2 = r$. (Every atom is join irreducible, but the converse is not true in general.) Let $R = \{r_0, r_1, \dots, r_k\}$ be all join irreducible elements in L . For every $u \in L$ denote $R_u = \{r \in R, r \leq u\}$, and note that $R_{u \wedge v} = R_u \cup R_v$. Define the following components¹¹:

$$\begin{aligned} \gamma_0 &= R(x_1), S_1(x_1, y_1) \\ \gamma_i &= S_i(x_{i+1}, y_{i+1}), S_{i+1}(x_{i+1}, y_{i+1}) \quad i = 1, k-1 \\ \gamma_k &= S_k(x_k, y_k), T(y_k) \end{aligned}$$

Consider the sentences Φ and Ψ below:

$$\Phi = \bigvee_{u < \hat{1}} \bigwedge_{r_i \in R_u} \gamma_i \quad \Psi = \bigwedge_{u < \hat{1}} \bigvee_{r_i \in R_u} \gamma_i$$

Then both $\hat{\mathbf{L}}_D(\Phi)$ and $\hat{\mathbf{L}}_C(\Psi)$ satisfy the theorem. \square

Thus, any complete PTIME algorithm must decide whether or not to compute $\lambda(\hat{0})$, which amounts to deciding whether $\mu(\hat{0}, \hat{1}) = 0$.

5.2 Outline of the Completeness Proof

In this section we give an outline of the completeness proof and defer details to the Appendix.

THEOREM 5.7. *Suppose Φ is unsafe. Then there exists a rewriting $\Phi \xrightarrow{*} \varphi$ s.t. (a) φ has no separators, and (b) there exists a PTIME algorithm for evaluating $P_{\mathbf{A}}(\varphi)$ on probabilistic structure \mathbf{A} , with a single access to an oracle for computing $P_{\mathbf{B}}(\Phi)$.*

By definition, Φ is unsafe if there exists $\Phi \xrightarrow{*} \varphi$ for some sentence φ without separators. The theorem applies only to *some* rewriting, namely to those where all lattice rewritings $\Phi \rightarrow \varphi$ (third in Def. 5.1) are such that φ is a maximal unsafe element in the lattice.

Thus, the evaluation problem for φ can be reduced to that of Φ : in order to prove that unsafe sentences are hard, it suffices to prove that sentence without separators are hard. For that, we will further apply the rewrite rules, until every atom has at most two variables. But we may get stuck because we may be unable to find a level, as illustrated by:

$$\varphi = R(x, y), S(y, z) \vee R(x', y'), S(x', y')$$

If we list $\text{Vars}(\varphi)$ as x, x', y, y', z , each consecutive pair of variables is unifiable. This indicates that no level exists: if there were a level W , then W would need to include at least one variable, which implies that W contains *all* variables due to the unifications. This

¹⁰It rewrites to $R(a, x_1), S(a, x_1, y_1), S(a, x_2, y_2), T(a, y_2) \rightarrow R(a, x_1), S(a, x_1, y_1) \vee S(a, x_2, y_2), T(a, y_2)$.

¹¹That is, $\bigvee \gamma_i = h_k$.

contradicts the requirement that an atom may contain at most one variable in a level. While this sentence already has only two variables per atom, it illustrates where we may get stuck in trying to apply a rewriting.

To circumvent this, we transform the sentence as follows. Let $V = \text{Vars}(\varphi)$. A *leveling* is a function $l : V \rightarrow [L]$, where $L > 0$, s.t. for all $i \in [L]$, $l^{-1}(i)$ is a level. Conceptually, l partitions the variables into levels, and assigns an integer to each level. This, in turn, associates exactly one level to each relation attribute, since unifiable variables must be in the same level. We also refer to the pair φ, l as a *leveled sentence*.

Call a structure \mathbf{A} *leveled* if there exists a function $l : A \rightarrow [L]$ s.t. all constants that appear in a given relation attribute have the same level. This means that attributes associated with different levels cannot have any values in common. If one thinks of the structure A as a sentence by treating each constant as a variable and each tuple as an atom, then the structure is leveled iff the associated sentence is leveled.

PROPOSITION 5.8. *Let φ be a disjunctive sentence that has no separators. Then there exists $L > 0$ and a leveled sentence φ^L s.t. that φ^L has no separator and the evaluation problem of φ^L over L -leveled structures can be reduced in PTIME to the evaluation problem of φ .*

The proof is in the Appendix. We illustrate the main idea on the example above. We choose $L = 4$ and the leveled sentence φ becomes:

$$\varphi^L = R_{23}(x_2, y_3), S_{34}(y_3, u_4) \vee R_{12}(x_1, y_2), S_{23}(y_2, z_3) \vee R_{23}(x'_2, y'_3), S_{23}(x'_2, y'_3)$$

Note that φ^L still does not have a separator. We claim that φ and φ^L are equivalent over 4-leveled probabilistic structures: thus, hardness of φ^L on 4-leveled structures implies hardness of φ . We only need to apply the leveling construct once: if φ is leveled, if rewrite it as $\varphi \rightarrow \varphi'$ then φ' is also leveled. We prove:

THEOREM 5.9. *Suppose φ is leveled, and has no separator. Then there exists a rewriting $\varphi \xrightarrow{*} \varphi'$ s.t. φ' has no separator and every atom in φ' has at most two variables.*

We prove this result in the appendix. Note that here we must be allowed to use non-simple rewritings $\varphi \rightarrow \varphi[A/W]$, where W is not a separator (of course) and A has more than one constant. We show in the appendix examples where the theorem fails if one restricts A to have size 1.

Once we reach a sentence with at most two variables in each atom, completeness of the algorithm follows from the following result:

THEOREM 5.10. *Suppose φ at most two variables in each atom, and is leveled. If φ has no separator, then the weak counting problem for φ is hard for #P, even when the input structures are restricted to leveled structures.*

6. AN ALGORITHM USING CONDITIONALS AND DISJOINTNESS

Conditioning and disjointness are two important techniques in probabilistic inference. The first expresses the probability of some Boolean expression Φ as $P(\Phi) = P(\Phi \mid X) * P(X) + P(\Phi \mid \neg X) * (1 - P(X))$ where X is a Boolean variable. This basic technique can be applied to a variety of settings, going beyond probabilistic inference, for example in order to construct a Binary Decision Diagram [14], one of the most powerful techniques in model

checking. Olteanu [10] has shown recently that BDDs can be derived for any safe conjunctive sentence without selfjoins. A second popular technique is disjointness: if Φ and Ψ are exclusive probabilistic events, then $P(\Phi \vee \Psi) = P(\Phi) + P(\Psi)$.

We give an algorithm for evaluating a sentences Φ over probabilistic structures, that replaces the Mobius step of Algorithm 5.1 with a sequence of conditioning and disjointness steps. Conditioning is on an entire sentence φ , as opposed to a single Boolean variable. When the algorithm succeeds, it runs in PTIME in the size of the probabilistic structure. However, we also show that the algorithm is incomplete; in fact, we claim that no algorithm based on these two techniques only can be complete.

The problem we address is the following. Given $\Phi = \bigvee \varphi_i$, compute $P(\Phi)$ in a sequence of conditioning/disjointness steps, without using Mobius' inversion formula. The second step of Algorithm 5.1 (existential quantification based on independence) remains the same and is not repeated here. For reasons discussed earlier, that step requires that we have a CNF representation of the sentence, $\Psi = \bigwedge \varphi_i$, but both conditioning and disjointness operate on disjunctions. We use De Morgan's laws $P(\bigwedge \varphi_i) = 1 - P(\bigvee \neg \varphi_i)$. Thus, we some abuse of terminology we assume that our input is a D-lattice, although its elements are labeled with negations of disjunctive sentences.

We illustrate first with an example

Example 6.1 Consider the sentence in Example 5.4:

$$\begin{aligned}\Phi &= R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2) \vee R(x_3), T(y_3) \\ &= \varphi_1 \vee \varphi_2 \vee \varphi_3\end{aligned}$$

We illustrate here directly on the DNF lattice, without using the negation. (This works in our simple example, but in general one must start from the CNF, then negate.) The Hasse diagram of the DNF lattice is shown in Fig. 1. First, let's revisit Mobius' inversion formula:

$$P(\Phi) = P(\varphi_1) + P(\varphi_2) + P(\varphi_3) - P(\varphi_1\varphi_3) - P(\varphi_2\varphi_3)$$

The only unsafe sentence in the lattice is the bottom element of the lattice, where φ_1 and φ_2 occur together, but that disappears from the sum because $\mu(\hat{0}, \hat{1}) = 0$. We show how to compute Φ by conditioning on φ_3 . We denote $\bar{\varphi} = \neg \varphi$ for a formula φ :

$$\begin{aligned}P(\Phi) &= P(\varphi_3) + P((\varphi_1 \vee \varphi_2) \wedge \bar{\varphi}_3) \\ &= P(\varphi_3) + P((\varphi_1, \bar{\varphi}_3) \vee (\varphi_2, \bar{\varphi}_3)) \\ &= P(\varphi_3) + P(\varphi_1, \bar{\varphi}_3) + P(\varphi_2, \bar{\varphi}_3) \\ &= P(\varphi_3) + (P(\varphi_1) - P(\varphi_1, \varphi_3)) + (P(\varphi_2) - P(\varphi_2, \varphi_3))\end{aligned}$$

The expansion based on conditioning on φ_3 is given in the third line. Notice that, given that φ_3 is false, the events φ_1 and φ_2 become mutually exclusive: the third line applies the disjointness principle. We expand one more step, in order to complete the computation of the probability: this is the fourth line above. This last expansion may be replaced with a different usage, e.g. the construction of a BDD, not addressed in this paper.

Consider what would happen if we conditioned on φ_1 instead:

$$\begin{aligned}P(\Phi) &= P(\varphi_1) + P((\varphi_2 \vee \varphi_3) \wedge \bar{\varphi}_1) \\ &= P(\varphi_1) + P(\varphi_2 \wedge \bar{\varphi}_1) + P(\varphi_3 \wedge \bar{\varphi}_1)\end{aligned}$$

Now we are stuck, because the expression $\varphi_2 \wedge \bar{\varphi}_1$ is #P hard.

Algorithm 6.1 computes the probability of a DNF formula using conditionals and disjointness. The algorithm operates on a DNF lattice (which, recall, may represent the negation of a CNF sentence). The algorithm starts by minimizing the expression $\bigvee_i \varphi_i$:

remove all sentences φ_i for which there exists $j \neq i$ s.t. $\varphi_i \Rightarrow \varphi_j$. We have seen in Prop 2.4 that this corresponds to removing all elements that are not co-atomic from the DNF lattice L . Recall that $\Phi = \bigvee_{u < \hat{1}} \lambda(u)$.

Next, the algorithm chooses a particular sublattice E , called the *eraser lattice*, and conditions on the disjunction of all sentences in the lattice. We define E below: first we show how to use it. Denote u_1, \dots, u_k the minimal elements of $L - E$. For any subset $S \subseteq L$, denote $\Phi_S = \bigvee_{u \in S, u < \hat{1}} \lambda(u)$; in particular, $\Phi_L = \Phi$.

The *conditioning* and the *disjointness* rules give us:

$$\begin{aligned}P(\Phi_L) &= P(\Phi_E) + P(\Phi_{L-E} \wedge (\neg \Phi_E)) \\ &= P(\Phi_E) + \sum_{i=1, k} (\Phi_{[u_i, \hat{1}]} \wedge (\neg \Phi_E))\end{aligned}$$

We have used here the fact that, for $i \neq j$, the sentences $\Phi_{[u_i, \hat{1}]}$ and $\Phi_{[u_j, \hat{1}]}$ are disjoint given $\neg \Phi_E$. Finally, we do this:

$$P(\Phi_{[u_i, \hat{1}]} \wedge (\neg \Phi_E)) = P(\Phi_{[u_i, \hat{1}]}) - P(\Phi_{[u_i, \hat{1}] \wedge E})$$

where $[u_i, \hat{1}] \wedge E = \{u \wedge v \mid u \geq u_i, v \in E - \{\hat{1}\}\}$

This completes the high level description of the algorithm. We show now how to choose the eraser lattice, then discuss why the algorithm is incomplete.

6.1 Computing the Eraser Lattice

Fix a lattice (\hat{L}, \leq) . The set of zero elements, Z , and the set of z -atoms ZA are defined as follows:

$$\begin{aligned}Z &= \{z \mid \mu_L(z, \hat{1}) = 0\} \\ ZA &= \{a \mid a \text{ covers some element } z \in Z\}\end{aligned}$$

The algorithm reduces the problem of computing $P(\Phi_L)$ for the entire lattice L to computing $P(\Phi_K)$ for three kinds of sub-lattices K : E , $[u_i, \hat{1}] \wedge E$, and $[u_i, \hat{1}]$. The goal is to choose E to avoid computing unsafe sentences. We assume the worse: that every zero element $z \in Z$ is unsafe (if a non-zero element is unsafe then the sentence is hard). So our goal is: choose E s.t. for any sub-lattice K above, if z is a zero element and $z \in K$, then $\mu_K(z, \hat{1}) = 0$. That is, we can't necessarily remove the zeros in one conditioning step, but if we ensure that they continue to be zeroes, so they will eventually be eliminated.

The *join closure* of a subset $S \subseteq L$ is defined as $cl(S) = \{\bigvee_{u \in S} u \mid s \subseteq S\}$. Note that $\hat{0} \in cl(S)$. The join closure is a join-semilattice and is made into a lattice by adding $\hat{1}$ and defining $u \wedge v = \bigvee \{w \mid w \in cl(S), w \leq u, w \leq v\}$.

DEFINITION 6.2. Let L be a lattice. The *eraser lattice* $E \subseteq L$ is $E = \{\hat{1}\} \cup cl(Z \cup ZA)$.

The following three propositions, proved in the appendix, show that E has our required properties.

PROPOSITION 6.3. If $u \in L$ and $w \in [u, \hat{1}]$ then $\mu_{[u, \hat{1}]}(u, \hat{1}) = \mu_L(u, \hat{1})$.

PROPOSITION 6.4. For all $z \in Z$, $\mu_E(z, \hat{1}) = 0$.

PROPOSITION 6.5. Assume $\hat{0} \in Z$. Then, for every zero element $z \in L$ and for every $w \in L - E$ we have $\mu_{[\hat{0}, w]}(z, w) = 0$.

Example 6.6 Consider Example 6.1. The eraser lattice for Fig. 1 (a) is the join closure of $\{\hat{0}, (\varphi_1, \varphi_3), (\varphi_3, \varphi_2)\}$ and consists of this set, plus φ_3 and $\hat{1}$. Notice that this set is not co-atomic: in other words, when viewed as a sentence, it minimizes to φ_3 .

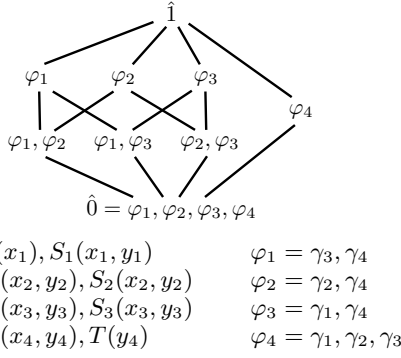


Figure 2: A lattice where $\mu(\hat{0}, \hat{1}) = 0$, yet is both atomic and co-atomic: there is no eraser here. This lattice corresponds to the D-lattice of concrete sentence given by Th. 5.6: $\Phi = \varphi_1 \vee \varphi_2 \vee \varphi_3 \vee \varphi_4$. Compare to h_3 in Sec. 4.2.

To get a better intuition on how conditioning works from a lattice-theoretic perspective, consider the case when $Z = \{\hat{0}\}$. In this case ZA is the set of atoms, and E is simply the set of all atomic elements; usually this is a strict subset of L , and conditioning partitions the lattice into E , $[u_i, \hat{1}] \wedge E$, and $[u_i, \hat{1}]$. When processing E recursively, the algorithm retains only co-atomic elements. Thus, conditioning works by repeatedly removing elements that are not atomic, then elements that are not co-atomic, until $\hat{0}$ is removed, in which case we have removed the unsafe sentence and we can proceed arbitrarily.

6.2 Incompleteness

Conditioning corresponds to repeatedly trimming the lattice to the atomic-, then to the co-atomic-elements, until $\hat{0}$ is removed. Proposition 2.4 implies that, if $\hat{0}$ is eventually removed this way, then $\mu(\hat{0}, \hat{1}) = 0$. But does the converse hold?

The answer is “no”, as shown by the lattice in Fig. 2. Here $\mu(\hat{0}, \hat{1}) = 0$, yet the lattice is both atomic and co-atomic. When computing the eraser E , one gets the entire lattice $E = L$, because every element is a join of all atoms. Removing elements that are not co-atomic doesn’t help either: all elements are co-atomic.

To make the example more concrete, recall from Th. 5.6 that there exists a sentence Φ that generates this lattice, where all elements are safe, except for $\hat{0}$ which is unsafe: the sentence is shown in the Figure. Thus, we have an example of a sentence that is in PTIME (simply use Mobius’ inversion formula), yet we cannot make any progress on it by applying conditioning or disjointness. Note that Algorithm 5.1 will first rewrite Φ into CNF, which happens to be a lattice isomorphic to that in Fig. 2.

7. CONCLUSIONS

We have proposed a simple, but quite non-obvious algorithm for computing the probability of an existential, positive sentence over a probabilistic structure. For every *safe* sentence, the algorithm runs in PTIME in the size of the input structure; every *unsafe* sentence is hard. Our algorithm relies in a critical way on Mobius’ inversion formula, which allows it to avoid attempting to compute the probability of sub-sentences that are hard. We have also discussed the limitations of an alternative approach to computing probabilities, based on conditioning and independence.

Acknowledgments We thank Chrisoph Koch and Paul Beame for pointing us (independently) to incidence algebras.

Algorithm 6.1 Compute the probability of Φ using conditionals and disjointness

Input: $\Phi = \bigvee_{i=1,m} \varphi_i$, $L = L_{DNF}(\Phi)$

Output $P(\Phi)$

```

1: Function Conditioning( $L$ )
2: If  $L$  has a single co-atom Then proceed with IndepStep
3: Remove from  $L$  all elements that are not co-atomic (Prop 2.4)
4: Let  $Z = \{u \mid u \in L, \mu_L(u, \hat{1}) = 0\}$ 
5: Let  $ZA = \{u \mid u \in L, u \text{ covers some } z \in Z\}$ 
6: If  $Z = \emptyset$  Then  $E := [u, \hat{1}]$  for arbitrary  $u$ 
7:   Else  $E :=$  the join-closure of  $Z \cup ZA$ 
8: If  $E = L$  then FAIL (unable to proceed)
9: Let  $u_1, \dots, u_k$  be the minimal elements of  $L - E$ 
10: Return Conditioning( $E$ ) +  $\sum_{i=1,k} \text{Cond1}(u_i, E)$ 
11:
12:   where Cond1( $u, E$ ) =
13:   Conditioning( $u$ ) + Conditioning( $([u, \hat{1}] \wedge E)$ )

```

8. REFERENCES

- [1] J. Boulos, N. Dalvi, B. Mandhani, S. Mathur, C. Re, and D. Suciu. Mystiq: A system for finding more answers by using probabilities. In *SIGMOD*, 2005. system demo.
- [2] N. Creignou and M. Hermann. Complexity of generalized satisfiability counting problems. *Inf. Comput.*, 125(1):1–12, 1996.
- [3] Nadia Creignou. A dichotomy theorem for maximum generalized satisfiability problems. *J. Comput. Syst. Sci.*, 51(3):511–522, 1995.
- [4] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, Toronto, Canada, 2004.
- [5] N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, pages 293–302, 2007.
- [6] N. Dalvi and D. Suciu. Management of probabilistic data: Foundations and challenges. In *PODS*, pages 1–12, Beijing, China, 2007. (invited talk).
- [7] Adnan Darwiche. A differential approach to inference in bayesian networks. *Journal of the ACM*, 50(3):280–305, 2003.
- [8] Kevin H. Knuth. Lattice duality: The origin of probability and entropy. *Neurocomputing*, 67:245–274, 2005.
- [9] G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. *J. Comput. System Sci.*, 73(3):507–534, 2007.
- [10] Dan Olteanu and Jiewen Huang. Secondary-storage confidence computation for conjunctive queries with inequalities. In *SIGMOD*, pages 389–402, 2009.
- [11] Dan Olteanu, Jiewen Huang, and Christoph Koch. Sprout: Lazy vs. eager query plans for tuple-independent probabilistic databases. In *ICDE*, pages 640–651, 2009.
- [12] Yehoshua Sagiv and Mihalis Yannakakis. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM*, 27:633–655, 1980.
- [13] Richard P. Stanley. *Enumerative Combinatorics*. Cambridge University Press, 1997.
- [14] Ingo Wegener. BDDs—design, analysis, complexity, and applications. *Discrete Applied Mathematics*, 138(1-2):229–251, 2004.

APPENDIX

A. SOME PROOFS

Proof of Proposition 2.4

PROPOSITION A.1. [13, pp.159, exercise 30] Let (\hat{L}, \leq) be a finite lattice. A mapping $x \rightarrow \bar{x}$ on \hat{L} is called a closure if for all $x, y \in \hat{L}$: (a) $x \leq \bar{x}$, (b) if $x \leq y$ then $\bar{x} \leq \bar{y}$, and (c) $\bar{\bar{x}} = \bar{x}$. A closed element is an element x s.t. $x = \bar{x}$. Denote \bar{L} the subset of closed elements. Then:

$$\mu_{\bar{L}}(\bar{x}, \bar{y}) = \sum_{z \in \bar{L}: \bar{z} = \bar{y}} \mu_{\hat{L}}(\bar{x}, z)$$

COROLLARY A.2. Let $\bar{L} \subseteq \hat{L}$ be a subset that is closed under meet. If all coatoms are in \bar{L} then for all $u \in \bar{L}$, $\mu_{\bar{L}}(u, \hat{1}) = \mu_{\hat{L}}(u, \hat{1})$.

PROOF. The mapping $x \rightarrow \bar{x} = \bigwedge_{y \in \bar{L}: y \leq x} y$ is a closure. The proposition follows from the fact that the only element z s.t. $\bar{z} = \hat{1}$ is $z = \hat{1}$ (because all coatoms are closed). \square

PROOF. (of Proposition 2.4) (1) See [13]. (2) Apply Corollary A.2 to the set of co-atomic elements. \square

Proof of Proposition 4.5

PROOF.

- Let $V = \text{Vars}(\varphi)$ and construct a graph on V such that (u, v) is an edge whenever some atom of φ imposes the constraint $u < v$ based on ranking. This graph is acyclic because there exists a valuation h from φ to a ranked structure, which means $h(u) < h(v)$ for all edges (u, v) . Let $x_1, \dots, x_{|V|}$ be a topological ordering of V , i.e., such that $i < j$ for all edges (x_i, x_j) . Replace each x_i with i in φ , and let \mathbf{A} be the ranked structure consisting of all atoms in φ after this mapping. Clearly $\mathbf{A} \models \varphi$, hence $\mathbf{A} \models \varphi'$: the latter gives a valuation $\varphi' \rightarrow \mathbf{A}$, which, composed with the mapping $i \mapsto x_i$, gives a homomorphism $\varphi' \rightarrow \varphi$.
- Standard argument, omitted.
- Assume for contradiction that there exists p , s.t. $\forall i$, it is not the case that $\varphi_i \Rightarrow \varphi'_p$. Let \mathbf{A}_i be a structure s.t. $\mathbf{A}_i \models \varphi_i$ but $\mathbf{A}_i \not\models \varphi_p$. The active domain of \mathbf{A}_i is unconstrained by φ_i because the sentence is in reduced form, and hence contains no constants. Hence we may assume w.l.o.g. that for $i_1 \neq i_2$, the structures \mathbf{A}_{i_1} and \mathbf{A}_{i_2} have disjoint active domains. Define $\mathbf{A} = \bigcup_i \mathbf{A}_i$. Then $\mathbf{A} \models \bigvee \varphi_i$, implying that $\mathbf{A} \models \bigvee \varphi'_j$. In particular, $\mathbf{A} \models \varphi'_p$, hence there exists a valuation $\varphi'_p \rightarrow \mathbf{A}$. Since the active domains of each \mathbf{A}_i are disjoint, its image must be contained in a single \mathbf{A}_i , contradicting the fact that $\varphi_i \not\Rightarrow \varphi'_p$.

\square

Proof of Theorem 5.7

PROOF. By induction on the length of unsafety proof. (1) Assume φ is unsafe because $\varphi \rightarrow \varphi[a/V]$ and $\varphi[a/V]$ is unsafe. Suppose we are given an input structure \mathbf{A} . Remove from the input structure \mathbf{A} all tuples that do not have the constant a on the position of the separator variable: this does not affect the value $P(\varphi[a/V])$. Denote \mathbf{B} the new structure. Thus, $P_A(\varphi[a/V]) = P_B(\varphi)$. (2) Assume $\varphi = \varphi_0 \vee \varphi_1$ and φ_1 is unsafe. Let \mathbf{A} be a structure. Remove from \mathbf{A} all ground tuples that occurs in φ_0 ; this does not affect $P(\varphi_1)$, because the alphabet is ranked, hence φ_0 and φ_1 are independent; call \mathbf{B} the new structure. Then $P_A(\varphi_1) = P_B(\varphi)$. (3) Assume $\Phi \rightarrow \varphi$, where $\varphi = \lambda(u)$, for $u \in \mathbf{L}_C$ s.t. $\mu(u, \hat{1}) \neq 0$.

Here we will choose u s.t. it is a maximal element in the lattice with this property. In other words, for all $v > u$, $\lambda(v)$ is safe. That is, we choose a particular rewriting from Φ into a sentence without separator. Let \mathbf{A} be an input structure. We construct a new structure \mathbf{B} by adding some more tuples to \mathbf{A} , as follows. Write $\varphi = \bigvee \gamma_i$. Let $v \in L$ be such that $u \not\leq v$, and denote $\varphi' = \lambda(v) = \bigvee_j \gamma'_j$. Since $\varphi' \not\Rightarrow \varphi$, there exists a component γ'_j s.t. for any i , $\gamma'_j \not\Rightarrow \gamma_i$. Let \mathbf{A}_v be a structure obtained from γ'_j by substituting a fresh constant for each variable; that is, $\mathbf{A}_v \models \gamma'_j$. Moreover, set the probabilities of all these tuples to 1. We claim that $P_A(\varphi) = P_{\mathbf{A} \cup \mathbf{A}_v}(\varphi)$. Suppose a component γ_i of φ is true on some substructure of $\mathbf{A} \cup \mathbf{A}_v$. Since γ_i is a component, it must be true in either a substructure of \mathbf{A} or in \mathbf{A}_v : to prove this we use that fact that the only constants shared between \mathbf{A} and \mathbf{A}_v are those in C (the constants in Φ), and none of the variables in γ_i can be mapped to C (because the vocabulary is ranked w.r.t. C), hence, since γ_i is a component it is mapped either entirely to one or the other. But γ_i cannot be mapped to \mathbf{A}_v , because that would mean that there exists a homomorphism $\gamma_i \rightarrow \gamma'_j$, implying $\gamma'_j \Rightarrow \gamma_i$. Thus, we can add to \mathbf{A} all structures \mathbf{A}_v for all v s.t. $u \not\leq v$, without affecting the probability of φ . Denote \mathbf{B} the resulting structure. We have: $P(\varphi)$ can be reduced in PTIME to $P(\Phi)$, because $P_B(\Phi) = P_B(\bigwedge_{v \in L} \lambda(v)) = P_B(\bigwedge_{v \geq u} \lambda(v))$ because for all $v \not\geq u$, $\lambda(v)$ is true on the substructure \mathbf{A}_v , and all those tuples have probability 1. Mobius' inversion formula gives us gives us $P_B(\Phi) = -\sum_{v \geq u} \mu(v, \hat{1}) P_B(\lambda(v))$. For $v = u$, we have $\mu(u, \hat{1}) \neq 0$ and $P_B(\lambda(u)) = P_A(\lambda(u)) = P_A(\varphi)$. For $v > u$ we have that $\lambda(v)$ is safe, and therefore we can compute $P_B(\lambda(v))$ in PTIME using Algorithm 5.1. This gives us $P_A(\varphi)$. \square

Proof of Prop. 5.8.

We start with some definitions that we will use in subsequent proofs. Fix a positive sentence Φ (or a disjunctive sentence). Two distinct variables x, y in Φ are *co-occurring* if there exists an atom that contains both x and y . Two pairs of variables (x, y) and (x', y') are *unifiable* if there exists two unifiable atoms r and r' such that r contains x, y on the same positions as r' contains x', y' .

DEFINITION A.3. The unification graph $G(\Phi)$ of a positive sentence Φ is the undirected graph where the nodes are pairs of co-occurring variables (x, y) , and there is an edge between any two unifiable pairs of variables (x, y) and (x', y') .

For a variable x denote $\text{atoms}(x)$ the set of all atoms that contain¹² x . Given two co-occurring variables x, y , we denote $x \gg y$ if $\text{atoms}(x) - \text{atoms}(y) \neq \emptyset$. Call a variable x a *root variable*¹³ if there is no variable y s.t. $x \ll y$.

DEFINITION A.4. [5] An inversion is a path $(x_0, y_0), \dots, (x_k, y_k)$ in $G(\Phi)$ s.t. $x_0 \gg y_0$ and $x_k \ll y_k$. We call k the length of the inversion. A root inversion is an inversion where all variables other than y_0 and x_k are root variables.

An inversion of length 0 is a pair (x_0, y_0) where $x_0 \gg y_0$ and $x_0 \ll y_0$: since $x_0 = x_k$, any inversion of length 0 is also a root inversion. The sentences h_k are canonical examples of sentences having root inversions of length k , for $k = 0, 1, 2, \dots$:

$$h_0 = R(x_0), S_1(x_0, y_0), T(y_0)$$

¹²If $x \in \text{Var}(\varphi_i)$ then $\text{atoms}(x) \subseteq \text{Atoms}(\varphi_i)$.

¹³This is consistent with the earlier definition of a root variable: when φ_i is a single component then x occurs in all atoms iff there is no variable y s.t. $x \ll y$.

$$\begin{aligned}
h_1 &= R(x_0), S_1(x_0, y_0) \vee S_1(x_1, y_1), T(y_1) \\
h_2 &= R(x_0), S_1(x_0, y_0) \vee S_1(x_1, y_1), S_2(x_1, y_1) \vee S_2(x_2, y_2), T(y_2) \\
&\dots \\
h_k &= R(x_0), S_1(x_0, y_0) \vee S_1(x_1, y_1), S_2(x_1, y_1) \dots \vee S_k(x_k, y_k), T(y_k)
\end{aligned}$$

Each sentence has a root inversion from (x_0, y_0) to (x_k, y_k) .
The following are easily checked.

LEMMA A.5. (a) A root variable x is a maximal root variable iff it does not participate in any root inversion. (b) If a disjunctive sentence has at most two variables in each atom then it has a separator iff it has no root inversion.

We can now proceed to the proof of Prop. 5.8.

PROOF. (Sketch) Assume wlog that φ is in reduced form, and all relations are ranked. Let L be “large enough” (to be specified below). Let a be the largest arity of any relation name, and v be the maximum number of variables in any component of φ . Let $R(A_1, \dots, A_k)$ be a relation name. Consider all strictly increasing level functions $m : [k] \rightarrow [L]$ s.t. $m(k) - m(1) \leq v$. For each m denote R^m a fresh relational symbol leveled by m . For each component γ_i , denote M a mapping from each atom r to a mapping m for r ; denote γ_i^M the component obtained by replacing each atom a leveled one. Retain only the consistent levelings, i.e. where each variable occurring in multiple atoms is assigned the same level. Let $\varphi^L = \bigvee_{i, M} \gamma_i^M$. Clearly φ^L is equivalent to φ over L -leveled structures, in the following sense. For each L -leveled structure \mathbf{A}^L denote \mathbf{A} the structure where, for every relational symbol R , $R^A = \bigcup_m (R^m)^{A^L}$. Then $\mathbf{A} \models \varphi$ iff $\mathbf{A}^L \models \varphi^L$.

Next we show that φ^L has no separator. Let γ_i be a component that has no maximal variables. Let M be a level labeling of the relations in γ_i that places them “in the middle” of the range L . Since γ_i is connected and each relation has a spread of levels $\leq av$, it follows that the spread of M is bounded by a, v , and the size of γ_i . We use the fact that γ_i has no maximal variables to prove that γ_i^M also has no maximal variables. Assume for contradiction that γ_i^M has a maximal variable x . We use the fact that γ_i has no maximal variables to prove that x (which is a root variable) recursively unifies with a non-root variable. Indeed, x recursively unifies with some variable y in φ . This is indicated by some sequence of variables x, v_1, v_2, \dots, y where each consecutive pair is unifiable. Assume wlog that this list does not contain duplicates: otherwise, the portion between the duplicates could be removed. We trace the same sequence in φ^L , where only predicates with the same level labeling are allowed to unify (different label functions on the same R are treated like different relational symbols). We need to check that all the consecutive pairs are indeed unifiable in φ : the labels may be “pushed out of range”. However, each unification step between $\gamma_i^{M_i}$ and $\gamma_j^{M_j}$ can shift the range of M only by an amount that is bounded by the sizes of γ_i and γ_j . Moreover, since the sequence is has no duplicates, its length is bounded by the size of φ . Thus, by choosing L “large enough” we ensure that if we start at the middle of the range $[L]$ we will not get out of range while following the sequence. \square

Example A.6 Let:

$$\varphi = R(x, y), S(y, z) \vee R(x', y'), S(x', y') = \gamma_0 \vee \gamma_1$$

Suppose both $R(X, Y)$ and $S(X, Y)$ are ranked as $X < Y$. There is an inversion $(x, y), (x', y'), (y, z)$ that goes twice through the first component, γ_0 , and uses y twice, in two different positions. Leveling avoids this. To level this sentence, we choose $L = 4$, and

consider three leveling labels for both R and S : 12, 23, and 34. The sentence becomes:

$$\begin{aligned}
\varphi^L &= R_{23}(x_2, y_3), S_{34}(y_3, u_4) \vee \\
&R_{12}(x_1, y_2), S_{23}(y_2, z_3) \vee \\
&R_{23}(x'_2, y'_3), S_{23}(x'_2, y'_3)
\end{aligned}$$

And inversion is $(x_2, y_3), (x'_2, y'_3), (y_2, z_3)$.

Proof of Theorem 5.9

Thus, we will assume that φ is leveled. For each variable x , denote $l(x)$ its level. Whenever two atoms r and r' unify, every pair of variables that are equated are at the same level.

Denote:

$$\begin{aligned}
rt(i) &= \{l(x) \mid x \text{ is a root variable in } \varphi_i\} \\
g(l) &= \{i \mid rt(i) = \{l\}\} \\
v(l, i) &= \{x \mid x \in \text{Vars}(\gamma_i), l(x) = l\} \\
v(l) &= \bigcup_i v(l, i) \\
\text{UNIQUE} &= \{l \mid \exists i \in g(l)\}
\end{aligned}$$

A level is in the UNIQUE set if there exists a component γ_i that has a unique root variable, which is at that level.

Given a level l and a set of constants A , we denote:

$$\begin{aligned}
\gamma_i[A/l] &= \bigvee_{\theta: v(l, i) \rightarrow A} \gamma_i[\theta] \\
\varphi[A/l] &= \bigvee_i \gamma_i[A/l]
\end{aligned}$$

This is the sentence obtained by substituting variables at level l with some constant in A . For example, if $A = \{a, b, c\}$ and there are two variables x, y at level l (in the same component, or in different components), then $\varphi[A/l] = \varphi[a/x, a/y] \vee \varphi[a/x, b/y] \vee \dots \vee \varphi[c/x, c/y]$. If we restrict the class of leveled structures s.t. the level l contains only constants in A , then obviously $\varphi \equiv \varphi[A/l]$.

Suppose φ has no separator. Our theorem follows from the following two lemmas.

LEMMA A.7. Suppose φ has no separator, and let $l \notin \text{UNIQUE}$. Let A be a set of constants not occurring in φ s.t. $|A| \geq |v(l, i)|$ for all i . Then $\varphi[A/l]$ has no separator.

PROOF. Given a component γ_i and $\theta : v(l, i) \rightarrow A$, denote $\gamma_i[\theta]$ the conjunctive sentence obtained from γ_i by substituting each variable at level l with some constant in A . First, we note that $\gamma_i[\theta]$ is connected: indeed, if $i \notin g(l)$ then none of the root variables of γ_i gets substituted with a constant, and $\gamma_i[\theta]$ is connected. If $i \in g(l)$ then $|rt(i)| \geq 2$ meaning that γ_i has at least two root variables. At least one remains after substituting one root variable with a constant.

Consider now a $\theta : v(l, i) \rightarrow A$ that is injective: there exists such θ because $|A| \geq |v(l, i)|$. Then $\gamma_i[\theta]$ is non-redundant in the expression $\varphi[A/l]$. More precisely, there is no homomorphism $\gamma_j[\theta'] \rightarrow \gamma_i[\theta]$: this is because any such homomorphism gives rise to a homomorphism $\gamma_j \rightarrow \gamma_i$ contradicting the fact that φ is non-redundant.

Finally, we show that $\varphi[A/l]$ has no separator. Suppose it had a separator V' . Construct a separator V for φ as follows. For each component γ_i , let θ be any injective substitution $\theta : v(l, i) \rightarrow A$. Define the separator variable x_i for γ_i to be the same separator variable for $\gamma_i[\theta]$ (the two sentences are isomorphic and “the same variable” means up to this isomorphism). Note that this definition

is independent on our choice of θ . Indeed, if θ' is another injective substitution that agrees with θ on at least one variable x ($\theta(x) = \theta'(x)$) then $\gamma_i[\theta]$ and $\gamma_i[\theta']$ have two atoms that unify, hence their separator variables must unify, which means that they correspond to the same variable in γ_i . It is easy to check that the set $\{x_i \mid i = 1, \dots\}$ is indeed a separator for φ : if γ_i and γ_j unify, then there exists substitutions θ, θ' s.t. $\gamma_i[\theta]$ unifies with $\gamma_j[\theta']$, and therefore the separator variables must unify as well. \square

Let γ_i be a component with a single root variable x . Define the subcomponents of γ_i as follows. Construct a graph where the nodes are the atoms of γ_i and there exists an edge between two atoms if they share at least one variable other than x . Denote $sc(\gamma_i)$ the subcomponents of γ_i . One can write

$$\gamma_i \equiv \exists x. \sigma_1(x) \wedge \dots \wedge \sigma_m(x)$$

where any two subcomponents $\sigma_i(x)$ and $\sigma_j(x)$ share no other variables except x . Equivalently, the sentence $\gamma_i[a/x]$ consists of m connected components, $\sigma_i[a/x]$, for $i = 1, m$. Denote the following:

$$\begin{aligned} \gamma &\leq^b \gamma' && \text{if } \forall \sigma \in sc(\gamma). \exists \sigma' \in sc(\gamma'). \sigma' \Rightarrow \sigma \\ l &\leq^\# k && \text{if } \forall j \in g(k). \exists i \in g(l). \gamma_i \leq^b \gamma_j \end{aligned}$$

Here $\sigma' \Rightarrow \sigma$ means logical implication, and is equivalent to the existence of a homomorphism $\sigma \rightarrow \sigma'$.

PROPOSITION A.8. *Assuming that φ is non-redundant and all components are minimized: \leq is a partial order on $UNIQUE$.*

PROOF. Both \leq^b and $\leq^\#$ are standard powerdomain orders, and hence are transitive. We prove antisymmetry. Suppose $l, k \in UNIQUE$ and $l \leq^\# k \leq^\# l$. Then for all $p \in g(l)$ exists $j \in g(k)$ and further exists $i \in g(l)$ s.t. $\gamma_i \leq^b \gamma_j \leq^b \gamma_p$. This means that for all $\sigma \in sc(\gamma_i)$, $\exists \sigma' \in sc(\gamma_j)$ s.t. there is a homomorphism $h : \sigma \rightarrow \sigma'$; and there further exists $\sigma'' \in sc(\gamma_p)$ s.t. there exists a homomorphism $h' : \sigma' \rightarrow \sigma''$. Compose these two homomorphisms to get $h'' : \sigma \rightarrow \sigma''$. Let x be the root variable at level l in γ_i : clearly $h''(x) = x$, because any homomorphisms preserves levels, and there can't be another variable at level l in γ_p (since it would have to co-occur with the root variable). Let now σ range over $sc(\gamma_i)$: since all h'' agree on x and x is the unique root variable in γ_i (since $l \in UNIQUE$), it follows that we can stitch them together to get a homomorphism $f : \gamma_i \rightarrow \gamma_p$. Since φ was non-redundant, f must be an isomorphism and also $i = p$. It follows also that γ_i is isomorphic to γ_j , which implies $i = j$, contradiction since we assumed that their roots are on different levels. \square

LEMMA A.9. *Suppose $UNIQUE$ has at least three elements. Let $l \in UNIQUE$ be a maximal element w.r.t. the partial order $\leq^\#$. Let A be a set of constants s.t. $|A| \geq |v(l, i)|$ for all i . Then, if φ has no separator, then $\varphi[A/l]$ has no separator.*

PROOF. The proof extends the previous proof. Here we need to cope with the fact that, if $i \in g(l)$ then γ_i may have a unique root, and that is at level l ; hence, for every constant a , $\gamma_i[a/l]$ consists of more than one connected components, $\sigma_1, \sigma_2, \dots$. Transforming the sentence into CNF, it will have a set of coatoms where each coatom contains a single σ_i . We need to show that these do not make certain other components redundant.

Let $k \in UNIQUE$, $k \neq l$ and $j \in g(k)$. Suppose γ_j has a unique root, at level k . Consider a sentence $\gamma_i[\theta]$ in the substitution $\gamma_j[A/l] = \bigvee_{\theta} \gamma_j[\theta]$. This sentence is still connected because its root is at level k and we are only substituting at level l . Moreover, if θ is injective, then k remains the single root for $\gamma_i[\theta]$ (we

have shown this earlier). Assume $\gamma_i[\theta]$ is not redundant in $\varphi[A/l]$. Then if $\varphi[A/l]$ has a separator, it must be at level k . However, $UNIQUE$ has at least three elements, there will be a second such $k' \in UNIQUE$, $k' \neq l$ and $k' \neq k$. Arguing similarly, every separator of $\varphi[A/l]$ must be at level k' , proving that there cannot be a separator.

Thus, it remains to prove that at least one $\gamma_j[\theta]$ is non-redundant. We have shown it earlier to be non-redundant, but now we have some new subcomponents σ_i that could make it redundant. We want to show that there can be no homomorphism $\sigma_i \rightarrow \gamma_j[\theta]$.

Formally, let θ be an injective substitution $v(l, j) \rightarrow A$ and consider $\gamma_j[\theta]$. Let x be the root variable in γ_i (the level of x is l). Then $\sigma \in sc(\gamma_i)$ makes $\gamma_j[\theta]$ redundant if there exists some constant a and a homomorphism $\sigma[a] \Rightarrow \gamma_j[\theta]$. This corresponds to a homomorphism $h : \sigma \rightarrow \gamma_j$. The root of γ_j is on a different level from that of x , hence $h(x)$ is a non-root variable. It follows that the entire image of h must be contained in some sub-component of γ_j : $h : \sigma \rightarrow \sigma'$. Equivalently: $\sigma' \Rightarrow \sigma$.

Transform $\varphi[A/l]$ into CNF by apply the distributivity law to $\gamma_i[a/x] = \bigvee_{\sigma \in sc(\gamma_i)} \sigma[a/x]$. Each disjunct will contain a single σ . We need to show that in at least one such disjunct, σ does not make γ_j redundant. Taking the negation: γ_j is completely redundant if: for all $\sigma \in sc(\gamma_i)$, $\exists \sigma' \in sc(\gamma_j)$ s.t. $\sigma' \Rightarrow \sigma$. Equivalently: $\gamma_i \leq^b \gamma_j$.

We only need to have at least one γ_j with $j \in g(k)$ that is not completely redundant: this suffices to argue that one component in $\varphi[A/l]$ has a unique root at level k . The negation of this property is the following. We lose the entire level k if: $\forall j \in g(k), \exists i \in g(l)$ s.t. $\gamma_i \leq^b \gamma_j$. This is precisely $l \leq^\# k$. But this cannot happen because we have chosen l to be a maximal element in this order, and $k \neq l$.

It follows that, for all $k \in UNIQUE$, $k \neq l$, there will be a component in $\varphi[A/l]$ that has a single root, on level k . Since $UNIQUE$ has at least three elements, it follows that $\varphi[A/l]$ cannot have a separator. \square

This completes the proof of the theorem. We recap it, and summarize it.

Let φ be a disjunctive sentence without separators. Consider the following two rewriting:

Rule 1 If there exists $l \notin UNIQUE$ s.t. $v(l) \neq \emptyset$ (i.e. there are variables on level l), pick any set of constants A s.t. $|A| \geq |v(l, i)|$ for all i , and rewrite:

$$\varphi \longrightarrow \varphi[A/l]$$

Rule 2 If there are at least three distinct levels with variables in $UNIQUE$, then let l be any maximal level in the order $\leq^\#$. Pick any set of constants A s.t. $|A| \geq |v(l, i)|$ for all i , and rewrite:

$$\varphi \longrightarrow \varphi[A/l]$$

Rule 3 If neither rule 1 nor rule 2 apply, then stop: the sentence has at most two variables in each atom, and has no separator.

To get a better insight of the lemma we illustrate with a few examples.

Example A.10 In the examples below we always refer to the components as $\gamma_1 \vee \gamma_2 \vee \dots$. Further assume that the variables x, y, z are assigned to levels 1, 2, 3.

For a trivial illustration of rule 1, consider:

$$R(x), S(x, y, z) \vee S(x, y, z), T(y)$$

Then: $rt(1) = \{1\}$, $rt(2) = \{2\}$. In words: “the root of γ_1 is x , the root of γ_2 is y ”. Both γ_1 and γ_2 have unique roots, hence *UNIQUE* contains the levels 1, 2. The Rule 1 can only be applied to levels not in *UNIQUE*, which leaves level 3: we substitute z with one constant and obtain:

$$R(x), S(x, y, a) \vee S(x, y, a), T(y)$$

This has at most two variables in each atom and has no separator.

For a small variation, consider

$$= R(x), S(x, y, z) \vee S(x, y, z), T(y) \vee U(x, y, z), V(x, y, z)$$

Then: $rt(1) = \{1\}$, $rt(2) = \{2\}$, $rt(3) = \{1, 2, 3\}$. We can still apply Rule 1 to z , since the only component that has z as root has at two extra roots: in other words, *UNIQUE* = $\{1, 2\}$. After Rule 1 the sentence becomes:

$$= R(x), S(x, y, a) \vee S(x, y, a), T(y) \vee U(x, y, a), V(x, y, a)$$

For a more complex example, consider:

$$\begin{array}{l} U(x) \quad R(x, z), \quad S(x, y, z) \\ \vee \quad \quad \quad S(x, y, z), \quad T(y, z) \\ \vee \quad \quad \quad R(x, z), \quad \quad \quad T(y, z) \end{array}$$

Here $rt(1) = \{1\}$, $rt(2) = \{2, 3\}$ and $rt(3) = \{3\}$ (that is, $\gamma_1, \gamma_2, \gamma_3$ have roots $\{x\}$, $\{y, z\}$ and $\{z\}$ respectively). Levels 1 and 3 are in *UNIQUE*: level 1 appears as root in γ_1 which has the single root x ; level 3 appears as root in γ_3 which has it as the single root. But we can use level 2 (y) in rule 1, and obtain:

$$\begin{array}{l} U(x) \quad R(x, z), \quad S(x, a, z) \\ \vee \quad \quad \quad S(x, a, z), \quad T(a, z) \\ \vee \quad \quad \quad R(x, z), \quad \quad \quad T(a, z) \end{array}$$

This has at most two variables in each atom, and no separator.

Example A.11 We illustrate now the need for Rule 2. This requires a more complex example. Consider φ below:

$$\begin{array}{l} R(x, y_1, z'_2), S(x, y_1, z''_2), R(x, y'_2, z_1), S(x, y''_2, z_1) \\ \vee \quad R(x_1, y, z'_2), T(x_1, y, z''_2), R(x'_2, y, z_1), T(x''_2, y, z_1) \\ \vee \quad S(x_1, y'_2, z), T(x_1, y'_2, z), S(x'_2, y_1, z), T(x''_2, y_1, z) \end{array}$$

$\gamma_1, \gamma_2, \gamma_3$ have the unique roots x, y, z , so all three levels are in *UNIQUE*. We need to apply Rule 2, and that will split one component into two subcomponents. We compute $\leq^\#$, which is trivial here, since $\gamma_1, \gamma_2, \gamma_3$ are incomparable: hence each of them is maximal. Pick arbitrarily the variable z and substitute with three constants $A = \{a_1, a_2, a_3\}$ (because there are up to three z -variables, e.g. in γ_1 there are z'_2, z''_2, z_1). Then $\varphi[a/z] = \varphi_1 \wedge \varphi_2$ where:

$$\begin{aligned} \varphi_1 &= \bigvee_{i_1, i_2, i_3} R(x, y_1, a_{i_1}), S(x, y_1, a_{i_2}), R(x, y'_2, a_{i_3}), S(x, y''_2, a_{i_3}) \\ &\vee \bigvee_{j_1, j_2, j_3} R(x_1, y, a_{j_1}), T(x_1, y, a_{j_2}), R(x'_2, y, a_{j_3}), T(x''_2, y, a_{j_3}) \\ &\vee \bigvee_k S(x_1, y'_2, a_k), T(x_1, y'_2, a_k) \end{aligned} \quad \text{LEMMA A.13. } U \wedge E = \{u\} \wedge E$$

and similarly for φ_2 . Note that, when $i_1 = i_2 = i_3$ the sentence in the first line minimizes. However, when the three constants are distinct, then it doesn't minimize (it is isomorphic to γ_1).

Example A.12 Consider a new rule: “equate all cooccurring pairs of variables from two chosen levels l, k ”. In some cases this may

result in simpler rewritings. In the example above, we could chose levels 2 and 3 and equate all the y_j 's with all z_k 's, to obtain:

$$\begin{array}{l} R(x, y_1, y_1), S(x, y_1, y_1), R(x, y_2, y_2), S(x, y_2, y_2) \\ \vee \quad R(x_1, y, y), T(x_1, y, y), R(x'_2, y, y), T(x''_2, y, y) \\ \vee \quad S(x_1, y, z), T(x_1, y, z), S(x'_2, y, z), T(x''_2, y, z) \end{array}$$

(Technically we can't do this directly, since the structure is ranked, hence in $R(x, y, z)$ we have $x < y < z$; instead we would introduce functional dependencies $y \rightarrow z$ and $z \rightarrow y$ in the ranked structure, which for all practical purposes is like equating $y = z$.) Thus, we have arrived much easier to a sentence with at most two variables per atom and without separators. One wonders if Rule 2 can be replaced with this rule. However, this doesn't work as illustrated by the following sentence, with 6 components. We show only the first component, $\gamma_1 =$

$$R(x, y_1, z'_1), S(x, y_1, z''_1), T(x, y_1, z_2), R(x, y_2, z_3), S(x, y_2, z'_3), T(x, y_2, z''_3)$$

The hierarchy between variables is unique among levels: $x \gg y \gg z$, where x is the root, y denotes the level with y_1, y_2 and z the level with z'_1, \dots . Consider all six permutation of this ordering of the levels, to obtain 6 components. Each of x, y, z is the unique root in two of the components.

Lets examine first what happens if we apply rule 2 but choose $|A| = 1$. The order $\leq^\#$ is here the identity, so all three components are maximal, and Rule 2 allows us to set, say, level 3 to a constant a . For example setting $z = a$ in γ_1 shown above results in $\gamma_1[a/3] \equiv R(x, y, a), S(x, y, a), T(x, y, a)$, because of minimization. Furthermore the other 5 components are now redundant (they all have a homomorphism to $\gamma_1[a/3]$), and the entire sentence is equivalent to just this expression, which is a safe sentence. Thus, we cannot apply rule 2 with a single constant, but need to chose a set A with more constants.

Lets examine now what happens if we apply our new rule, equating two variables. This has the same effect: for example, after setting $y = z$, γ_1 minimizes to $R(x, y, z), S(x, y, z), T(x, y, z)$; this is also safe.

The only rewriting is Rule 2, but done correctly, by choosing more than one constant for z , in order to ensure that one combination does not minimize.

Proofs from Section 6

PROOF. (Of Prop. 6.4) Consider the lattice $[z, \hat{1}]$: we have¹⁴ $\mu_{[z, \hat{1}]}(z, \hat{1}) = \mu_L(z, \hat{1}) = 0$. The set $E \cap [z, \hat{1}]$ is closed under joins (because E is closed under joins) and contains all atoms, hence the claim follows from the dual of Corollary A.2. \square

PROOF. (Of Prop 6.5)

Let $U \subseteq L - E$ s.t. U has a smallest element $u \in U$. Denote

LEMMA A.13. $U \wedge E = \{u\} \wedge E$

PROOF. Here $U \wedge E$ denotes $\{u \wedge v \mid u \in U, v \in E - \{\hat{1}\}\}$. It suffices to how that forall $u' \in U$ and $v \in L - \{\hat{1}\}$, $u \wedge v = u' \wedge v$. **TO PROVE.** \square

To prove Prop 6.5 it suffices to note that, in the lattice $[z, w]$ the top element w is not a join of atoms. The claim follows from the dual of Proposition 2.4 (1).

¹⁴This is a standard fact in incidence algebras: $\mu_L(x, y)$ depends only on the sublattice $[x, y]$.

B. PROOF OF THE FACT THAT FORBIDDEN SENTENCES ARE HARD

We include here a proof of a simplified variant of Theorem 5.10. We will refer to sentences as queries in this section.

B.1 Forbidden Queries

Let q be a forbidden query over relations $\mathcal{R} = \{R, S_1, \dots, S_k, T\}$. We can represent q using a set $\{\gamma_1, \dots, \gamma_k\}$, where each γ_i is a subset of \mathcal{R} . The sets satisfy two properties, (i) no set is redundant, i.e., contained in another set, and (ii) there is a chain $\gamma_{i_1}, \dots, \gamma_{i_l}$ such that $R \in \gamma_{i_1}, T \in \gamma_{i_l}$ and each adjacent pairs intersect.

Before we prove the hardness of forbidden queries, let us study some of their properties which we will use in our hardness proof. Each forbidden query $q = \{\gamma_1, \dots, \gamma_k\}$ defines a DNF formula $\phi(q)$ over the vocabulary \mathcal{R} , where each γ_i is a clause. For $i, j \in \{0, 1\}$, let $n_{i,j}(q)$ denote the number of non-satisfying assignments of $\phi(q)$ when R is set to i and T is set to j .

As an example, consider $q = \{\{R, S_1\}, \{S_1, S_2\}, \{S_2, T\}\}$. Then, $n_{1,1}(q)$ is the number of non-satisfying assignments when $R = 1$ and $T = 1$. We see that S_1 has to be 0 (otherwise $\{R, S_1\}$ will evaluate to true). Similarly, S_2 has to be 0. Thus, there is only 1 assignment and so $n_{1,1}(q) = 1$. For $n_{1,0}$, S_1 has to be 0 and S_2 can either be 0 or 1. Hence, $n_{1,0} = 2$. Similarly $n_{0,1} = 2$. Finally, $n_{0,0} = 3$, since S_1, S_2 cannot be both 1, but otherwise can take values for any other combination. We observe that $n_{1,1}(q)n_{0,0}(q) \neq n_{1,0}(q)n_{0,1}(q)$. This is an important property that we will exploit in our hardness proofs. Define $\Delta(q) = n_{1,1}(q)n_{0,0}(q) - n_{1,0}(q)n_{0,1}(q)$. We will give a hardness result for any forbidden query q that has $\Delta(q) \neq 0$. However, note that there do exist forbidden queries with $\Delta(q) = 0$. We use the following reduction technique to handle such queries.

Reducing Forbidden Queries.

Given any query $q = \{\gamma_1, \dots, \gamma_k\}$, we can set one of the relations to be empty to reduce it to a new query q' , whose set representation consists of all the sets of q not containing S_i . If q' is also hard, then we can show hardness of q using q' . E.g., is q is given by

$$\{\{R, S_1\}, \{S_1, S_2\}, \{S_2, T\}, \{R, S_3, S_1\}, \{S_3, T\}\}$$

we can set S_3 to be empty to obtain $\{\{R, S_1\}, \{S_1, S_2\}, \{S_2, T\}\}$, whose hardness guarantees hardness of q . Similarly, we can set any relation S_i to be deterministic relation containing all possible tuples from the active domain to obtain a new query q' , where S_i is removed from all sets. E.g. if we have $\{\{R, S_1, S_2\}, \{S_1, T\}\}$, we can set S_2 to be deterministic to obtain $\{\{R, S_1\}, \{S_1, T\}\}$. We say that q is reducible to q' if we can obtain q' from q by setting a subset of relations to be empty and a subset of relations to be deterministic.

THEOREM B.1. *Any forbidden query q is reducible to a query q' with $\Delta(q') \neq 0$.*

The above result shows that it is enough to prove the hardness for queries with $\Delta \neq 0$. In the rest of this section, we will prove this result. First, we start with a simple result. Let $n_{0,*}(q)$ denote the number of non-satisfying assignments of q when T is set to 0. Similarly define $n_{*,0}(q)$ and $n_{*,*}(q)$ (which is simply the total number of non-satisfying assignments).

$$\text{LEMMA B.2. } \Delta(q) = n_{*,*}(q)n_{0,0}(q) - n_{*,0}(q)n_{0,*}(q).$$

This follows directly from definitions. Next we show the following result. Given four polynomial functions f, g, h, k , define

$\Delta(f, g, h, k) = fg - hk$. Define a weak assignment of a polynomial as any mapping of variables to the set $\{0, 1, 1/2\}$.

LEMMA B.3 (FOUR-FUNCTIONS LEMMA). *Let f, g, h, k be any four multi-linear such that $\Delta(f, g, h, k) = 0$ for all weak assignments. Then, $\Delta(f, g, h, k)$ is identically 0.*

PROOF. We induct on the total number of variables. If all the four functions are constant, then the lemma follows trivially. Otherwise let x be any variable. Since f is multi-linear, we can write

$$f = (1 - x)f_0 + xf_1$$

where $f_0 = f[x = 0]$ and $f_1 = f[x = 1]$. Similarly for g, h, k . Thus, $\Delta(f, g, h, k) =$

$$\begin{aligned} &= fg - hk \\ &= ((1 - x)f_0 + xf_1)((1 - x)g_0 + xg_1) - ((1 - x)h_0 + xh_1)((1 - x)k_0 + xk_1) \\ &= (1 - x)^2\Delta(f_0, g_0, h_0, k_0) + x^2\Delta(f_1, g_1, h_1, k_1) + \\ &\quad x(1 - x)[\Delta(f_0, g_0, h_0, k_1) + \Delta(f_1, g_1, h_1, k_0) - \Delta(f_1 - f_0, g_1 - g_0, h_1 - h_0, k_1)] \end{aligned}$$

The last equation can be verified by expanding the terms. Now, for every weak assignment, $\Delta(f_0, g_0, h_0, k_0) = 0$. To see this, set $x = 0$ in the above equation and note that along with $x = 0$, this gives a weak assignment of f, g, h, k for which $\Delta(f, g, h, k)$ is 0. Similarly, by setting $x = 1$, we get that $\Delta(f_1, g_1, h_1, k_1)$ is 0 for all weak assignments. Finally, by setting $x = 1/2$ in the above equation and using the fact that both $\Delta(f_0, g_0, h_0, k_0)$ and $\Delta(f_1, g_1, h_1, k_1)$ are 0 for a weak assignment, we get that $\Delta(f_1 - f_0, g_1 - g_0, h_1 - h_0, k_1 - k_0)$ is 0 for any weak assignment. By induction, each of the three Δ functions are identically 0. Hence, $\Delta(f, g, h, k)$ is identically 0. \square

Now given a forbidden query q with the corresponding DNF formula $\phi(q)$ over \mathcal{R} , define a variable x_W for each $W \in \mathcal{R}$, and let $p(\phi(q))$ be the polynomial that gives the probability of q being false when the variable W is made true with probability x_W . We see that $p(\phi(q))$ is a multi-linear polynomial, where there is a monomial for each non-satisfying assignment of $\phi(q)$ with a factor x_W if W is true and $(1 - x_W)$ if W is false.

Let $a \equiv \phi(q)$, $b = \bar{x}_R$, $c = \bar{x}_T$ and let $f_q = p(a)$, $g_q = p(a \wedge b \wedge c)$, $h_q = p(q \wedge a)$ and $k_q = p(a \wedge c)$. By the above lemma, if $\Delta(f_q, g_q, h_q, k_q)$ is not identically 0, then there is some weak assignment for this the quantity is non-zero. Consider this assignment π . Let q' be the query obtained from q by making all the 0 variables of π deterministic and all the 1 variables of π deterministic. Then, we get that $\Delta(f'_q, g'_q, h'_q, k'_q) \neq 0$ when all the variables are set to 1/2. But this is precisely equal to $2^{\#vars(q')} (n_{*,*}(q')n_{0,0}(q') - n_{*,0}(q')n_{0,*}(q'))$. By Lemma B.2, this would imply $\Delta(q') \neq 0$. Thus, to prove Theorem B.1, all we need to show is that $\Delta(f_q, g_q, h_q, k_q)$ is not identically 0. For this we use the following result

THEOREM B.4. [9] *If $\Delta(a, a \wedge b \wedge c, a \wedge b, a \wedge c)$ is identically 0, then we can partition the variables into S_1 and S_2 such that $a = a_1 \wedge a_2$, $a \wedge b = a_1 \wedge b_2$, $a \wedge c = a_2 \wedge c_2$, and a_i, b_i, c_i only refer to variables in S_i .*

For the above result we can show that $\Delta(f_q, g_q, h_q, k_q)$ is not identically 0. If not, we can partition \mathcal{R} into S_1 and S_2 such that $a = a_1 \wedge a_2$, i.e. S_1 and S_2 are not connected in q , $a \wedge b = a_1 \wedge b_2$, which implies that $R \in S_1$ and $a \wedge c = a_2 \wedge c_2$, which implies that $T \in S_2$. This contradicts the definition of a forbidden query. Hence, we prove Theorem B.1.

B.2 Hardness of Queries With $\Delta \neq 0$

Let q be a query over relations R, S_1, \dots, S_k, T with $\Delta(q) \neq 0$. The basic unit of our construction is a *link*, as defined below. Let a, b be constants. Let c be a unique constant. A *link* between a and b is the following structure:

$$L(a, b) \equiv R(a), S_1(a, c), \dots, S_k(a, c), T(b), S_k(b, c), \dots, S_1(b, c), R(b) \quad \text{COROLLARY B.7. } \alpha(v)x(u, v) + y(u, v) = 1.$$

A *multi-link* between a and b of multiplicity v , denoted by $L_v(a, b)$, is simply a collection of v links between a and b . Note that each link is created by using a new unique constant. Thus, $L_1(a, b)$ is simply $L(a, b)$. Also observe that multi-link is a symmetric structure, i.e. $L_v(a, b) \equiv L_v(b, a)$. Let $R(a)$ and $R(b)$ denote the *end-points* of the link. Finally, a *chain* between a and b is constructed by creating links between a and b , i.e., for unique constants c_1, \dots, c_u , a chain of length u and multiplicity v is given by

$$C_{u,v}(a, b) \equiv L_v(a, c_1), L_v(c_1, c_2), \dots, L_v(c_{u-1}, c_u), L_v(c_u, b) \quad (9)$$

Given a bipartite monotone Boolean formula ϕ , the construction involves simply creating a chain between each pairs of variables that define a clause. Before we describe this construction, let us study some properties of links and chains, specifically evaluation of forbidden queries over links and chains.

In the rest of the section, we assume that q is a fixed query and all the notations are w.r.t this query. For $i = 0, 1, 2$, let $n_i(v)$ denote the number of possible worlds of $L_v(a, b)$ that *do not* satisfy q when exactly i of its end-points are false (i.e. not included in the possible world). For $i = 1$, it does not matter whether $R(a)$ is false or $R(b)$ is false, since links are symmetric. So $n_i(v)$ is well-defined.

LEMMA B.5. For $i = 0, 1, 2$, $n_i(v) = n_i^v(1)$.

PROOF. The only tuples that two different links between a and b share are the end-points $R(a)$ and $R(b)$. Further, any mapping of q to a multi-link is completely contained in one of the links. So once the truth assignment for the end-points is fixed, each of the v links can be independently counted. \square

Let $n_i(u, v)$ denote the number of possible worlds of a chain $C_{u,v}(a, b)$ of length u and multiplicity v . Again, since chains are symmetric, $n_i(u, v)$ is well-defined.

LEMMA B.6. We have the following recurrence:

$$\begin{aligned} n_2(u, v) &= n_2(v)n_2(u-1, v) + n_1(v)n_1(u-1, v) \\ n_1(u, v) &= n_2(v)n_1(u-1, v) + n_1(v)n_0(u-1, v) \\ &= n_1(v)n_2(u-1, v) + n_0(v)n_1(u-1, v) \\ n_0(u, v) &= n_0(v)n_0(u-1, v) + n_1(v)n_1(u-1, v) \end{aligned}$$

PROOF. We will only prove the first equation. Others have similar proof. Consider the chain $C_{u,v}(a, b)$ as defined in Eq. (9). In $n_2(u, v)$, we look at possible worlds where $R(a)$ and $R(b)$ are true. We group the possible worlds into two sets: where $R(c_1)$ is true and where $R(c_1)$ is false. Observe that once we fix the assignment of $R(c_1)$, the events that q is true on $L_v(a, c_1)$ is independent of the event that q is true on the rest of the chain.

For the case when $R(c_1)$ is true, there are $n_2(v)$ ways to pick a possible world of $L_v(a, c_1)$ that do not satisfy q . Also, there are $n_2(u-1, v)$ possible worlds of the rest of the chain where q is false. Thus, there are $n_2(v)n_2(u-1, v)$ worlds in the first case. Similarly, there are $n_1(v)n_1(u-1, v)$ worlds in the second case. This proves the first equation. \square

Define the following:

$$\begin{aligned} \alpha(v) &= (n_0(v) - n_1(v))/n_2(v) \\ x(u, v) &= n_1(u, v)/n_0(u, v) \\ y(u, v) &= n_2(u, v)/n_0(u, v) \end{aligned}$$

PROOF. Follows directly from the definitions and Lemma B.6 by using the two alternate expressions for $n_1(u, v)$ and setting $u \leftarrow u+1$. \square

LEMMA B.8. If q is a forbidden query with $\Delta(q) \neq 0$. Then $\alpha(v)$ is not a constant function. Also, for any fixed v , $x(u, v)$ and $y(u, v)$ are not constant functions.

PROOF. We have $\alpha(v) = (n_0(v) - n_1(v))/n_2(v)$, which, by Lemma B.5, equal $(n_0^v(1) - n_1^v(1))/n_2^v(1)$. This is not a constant iff $n_0(1) \neq n_1(1)$. Now, since q is a monotone query, its negation is a anti-monotone query. Hence, $n_0(1) \geq n_1(1)$. To show that the inequality is strict, we need to prove that there is at least one possible world for S_1, S_2, \dots, S_k, T such that if both the end points $R(a)$ and $R(b)$ are false than q is false, but it one end-point is true than q is true. So consider any set K of q containing R . Make all the S_i in K true and everything else false. This makes the query false (because if this made the query true, then we would have a set properly contained in K , which is not possible in forbidden queries). Setting any of the end-points 1 makes the query true. This shows $n_0(1) > n_1(1)$.

To show that $x(u, v)$ and $y(u, v)$ are not constant functions of u , let us solve the system of recurrence equations in Lemma B.6.

$$\begin{aligned} n_1(u, v) &= n_2(v)n_1(u-1, v) + n_1(v)n_0(u-1, v) \\ n_0(u, v) &= n_0(v)n_0(u-1, v) + n_1(v)n_1(u-1, v) \end{aligned}$$

Denoting $n_1(u, v)$ by $f(u)$ and $n_0(u, v)$ by $g(u)$, we can write the equations in matrix form as below:

$$\begin{bmatrix} f(u) \\ g(u) \end{bmatrix} = \begin{pmatrix} n_2(v) & n_1(v) \\ n_1(v) & n_0(v) \end{pmatrix} \begin{bmatrix} f(u-1) \\ g(u-1) \end{bmatrix}$$

The solution of the recurrence is given by

$$\begin{aligned} f(u) &= A\lambda_1^u + B\lambda_2^u \\ g(u) &= C\lambda_1^u + D\lambda_2^u \end{aligned}$$

where λ_1 and λ_2 are the eigenvalues of the matrix. It is easy to check that $x(u, v) = f(u)/g(u)$ is not a constant function of u iff both λ_1 and λ_2 are non-zero and both of them are not equal to 1. The eigenvalues are the roots of the equation

$$(\lambda - n_0(v))(\lambda - n_2(v)) - n_1^2(v) = 0$$

Both the roots are non-zero iff $n_0(v)n_2(v) \neq n_1^2(v)$, which is equivalent to $n_0(1)n_2(1) \neq n_1^2(1)$. Now,

$$\begin{aligned} n_0(1) &= n_{00}(q)n_{00}(q) + n_{01}(q)n_{10}(q) \\ n_1(1) &= n_{10}(q)n_{00}(q) + n_{10}(q)n_{11}(q) \\ n_2(1) &= n_{10}(q)n_{01}(q) + n_{11}(q)n_{11}(q) \end{aligned}$$

After substituting, we get $n_0(1)n_2(1) - n_1^2(1) = \Delta(q) \neq 0$. Also, both the eigenvalues cannot be 1, as this would imply

$$\begin{aligned} n_0(v) + n_2(v) &= 2 \\ n_0(v)n_2(v) - n_1^2(v) &= 1 \end{aligned}$$

Since each of these are positive integers, the only solution to this is $n_0(v) = n_2(v) = 1, n_1(v) = 0$, which cannot hold because $n_1(v) \geq n_2(v)$ because of the anti-monotonic property. \square

THEOREM B.9. *The counting version of q is #P-hard.*

PROOF. Proof is by reduction from #PP2DNF. Consider a bipartite monotone Boolean formula $\phi = \bigvee x_i y_j$. Let N be the total number of clauses. We create an instance for q as follows.

For each $x_i y_j$ in ϕ , create a chain of length u and multiplicity v between x_i and y_j . Let $\mathcal{A}(\phi)$ be the resulting structure. Let $F(u, v)$ be the number of possible worlds of $\mathcal{A}(\phi)$ that *do not* satisfy q , as a function of u, v .

Let $m(a, b)$ be the number of assignments of ϕ such that a clauses have both variable false and b clauses have exactly 1 variable false (and hence, $N - a - b$ clauses have both variables true). Consider a mapping from the possible worlds of $\mathcal{A}(\phi)$ to assignments of ϕ , where a variable x is 0 if the corresponding tuple $R(x)$ is present in the possible world.

Given an assignment of ϕ where a clauses have both variables false and b clauses have 1 variable false, the number of possible worlds of $\mathcal{A}(\phi)$ that map to this assignment and *do not* satisfy q is given by $n_2^a(u, v) n_1^b(u, v) n_0^{N-a-b}(u, v)$. Thus, we have

$$F(u, v) = \sum_{a,b} m(a, b) n_2^a(u, v) n_1^b(u, v) n_0^{N-a-b}(u, v)$$

Let $\alpha (= \alpha(v))$, $x (= x(u, v))$ and $y (= y(u, v))$ be as defined above. Then, we have

$$F(u, v) n_0^{-N}(u, v) = \sum_{a,b} m(a, b) x^a y^b = \sum_{a,b} m(a, b) x^a (1 - \alpha x)^b$$

The *R.S.H* can be viewed as a polynomial in x . By fixing v (and thus α) and varying u , we can plug in different values of x to get all the co-efficients of the polynomial. The leading co-efficient is $\sum_{a+b=N} m(a, b) (-\alpha)^b$. Now, we can view this as a polynomial in α . By varying v , we can plug in different values of α to get all the co-efficients. The sum of the co-efficients is precisely the number of non-satisfying assignments of ϕ . Thus, we can use q to count the number of satisfying assignments of any ϕ . \square